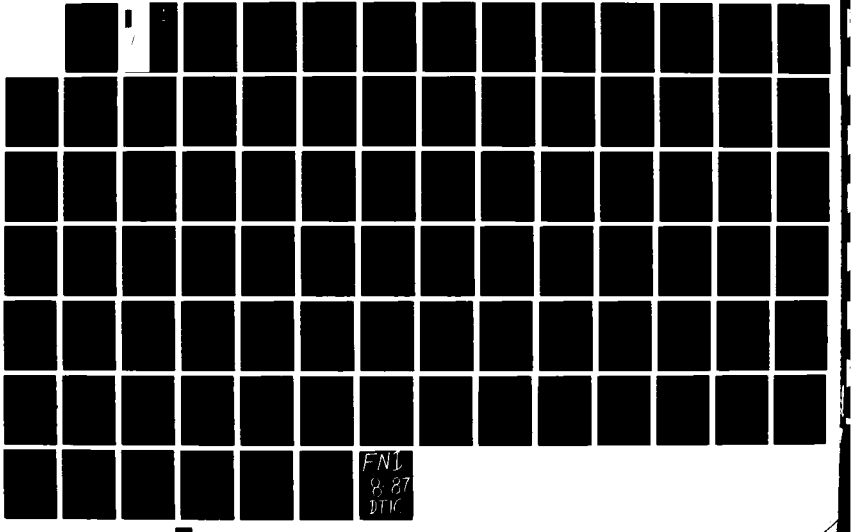
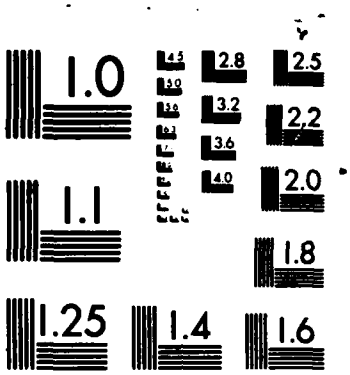
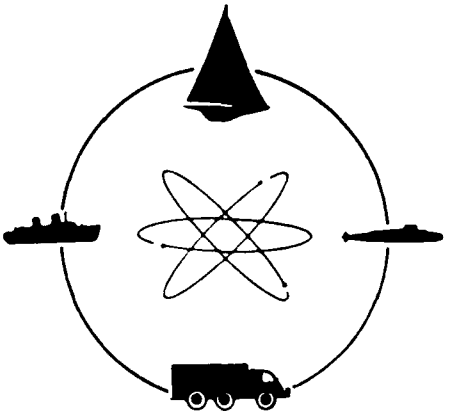
AD-A182 801 ALTERNATIVE METHOD OF CALCULATION OF EFFECTIVE WAKE WITH RADIAL AND TANGENTIAL SHEARS(U) STEVENS INST OF TECH HOBOKEN NJ DAVIDSON LAB G MCKEE 31 JUL 86 1/1

UNCLASSIFIED SIT-DL-86-9-2556 N00014-84-K-0071 F/G 20/4 NL





TR-2556



DAVIDSON LABORATORY

TECHNICAL REPORT SIT-DL-86-9-2556

July 1986

ALTERNATIVE METHOD OF CALCULATION OF EFFECTIVE WAKE WITH RADIAL AND TANGENTIAL SHEARS

by
Glenn McKee

Prepared for

Office of Naval Research
General Hydrodynamic Research Program

under

Contract N00014-84-K-0071

VENS INSTITUTE
TECHNOLOGY

TLE POINT STATION
EN, NEW JERSEY 07030

TR-2556

STEVENS INSTITUTE OF TECHNOLOGY
DAVIDSON LABORATORY
Castle Point Station, Hoboken, New Jersey 07030

TECHNICAL REPORT SIT-DL-86-9-2556

JULY 1986

ALTERNATIVE METHOD OF CALCULATION OF EFFECTIVE WAKE WITH RADIAL AND TANGENTIAL SHEARS

by

Glenn McKee

Prepared for

Office of Naval Research
General Hydrodynamic Research Program

under

Contract N00014-84-K-0071

APPROVED: P. Ward Brown

P. Ward Brown
Manager, Marine Craft
Development Group

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

ADA182801

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION / AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) SIT-DL-86-9-2556	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Stevens Inst. of Technology Davidson Laboratory 6b OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER, Code 1504 (1505)
6c. ADDRESS (City, State, and ZIP Code) Castle Point Station Hoboken, New Jersey 07030	7b. ADDRESS (City, State, and ZIP Code) Bethesda, Maryland 20084-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Sea Systems Command 8b OFFICE SYMBOL (If applicable) SEA 05R24	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Contract Number- N00014-84-K-0071
8c. ADDRESS (City, State, and ZIP Code) Washington, D.C. 20360	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 61153N PROJECT NO: SR 023 01 TASK NO: 23454 WORK UNIT ACCESSION NO: N/A

11 TITLE (Include Security Classification)
Alternative Method for Calculation of Effective Wake with Radial and Tangential Shears

12 PERSONAL AUTHOR(S)
Dr. Glenn McKee

13a. TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
FINAL	FROM 01/83 TO 03/86	860731	82

16 SUPPLEMENTARY NOTATION
Sponsored under the Naval Sea Systems Command General Hydromechanics Research (GHR) Program administred by the David W. Taylor Naval Ship R&D Center, Code 1504(1505), Bethesda, Maryland 20084-5000

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
20	04	

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
GHR Program, Shear Flow, Propulsion, Hydrodynamics, Computational

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The partial differential equation for the perturbed pressure field of a propulsor operating in a non-uniform flow was transformed into a simple form due to Goodman. Starting from the available solution for a flow with no shear, an iterative procedure was selected to derive a numerical solution for shear flow.

The equivalent integral equation based on solving Laplace's equation would involve the numerical calculation of a five-fold integral over an infinite computational domain for each iteration.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Mr. V.J. Monacella	22b TELEPHONE (Include Area Code) 202-227-1503 22c. OFFICE SYMBOL Code 1504/1505

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

(ABSTRACT CONTINUED)

An alternative integral equation was derived by considering the equation as an "inhomogeneous ordinary differential equation" in the longitudinal derivatives, and including the lateral derivatives in the forcing function. This method involves only one numerical integral per iteration, and handles the boundary conditions at infinity and the propulsor.

The numerical method worked when the lateral derivatives were zero, but successive iterations incoporating the lateral derivatives indicate a basic instability in the original partial differential equation.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

0. Introduction

The objective of this research was the development of a rational procedure for the determination of the spatial distribution of inflow to a propeller when operating in a non-axially symmetric hull wake. A knowledge of the various effective spatial harmonic amplitudes is primarily important in predicting both the inception and extent of blade cavitation. Secondarily,the alteration of the radial load distribution affects propeller efficiency as compared to its value as secured from open-water tests.

0.1. Derivation of Goodman's Equation

A propeller operating in the wake of a marine vehicle interacts with the wake thereby modifying the flow into the propeller from that of the nominal wake measured in the absence of the propeller. The resulting change in the velocity distribution at the propeller plane is due to the change in pressure induced by the propeller in the wake flow of the vehicle. The resulting inflow velocity distribution ("effective wake") which is experienced by the propeller is a result of the interaction of the propeller and the wake and it is different from the nominal velocity distribution (nominal wake) determined in the absence of the propeller.

Although the conventional procedure [1] of adjusting the nominal wake by a constant factor determined from resistance, self-propulsion, and open-water experiments provides a reasonable estimate of the average inflow to the propeller, it is unable to determine the actual radial distribution of this inflow. The detailed knowledge of this distribution is especially important in predicting powering and cavitation performance.

Goodman developed a linearized axial momentum theory for a lightly-loaded disc operating in an axially directed shear flow [2] He examined the effect of the vorticity contained in the shear flow experienced by the propeller on the optimum propeller characteristics. He found that the velocities induced by the propeller at its location were substantially altered due to the shear. He also extended his model to include the intensification of the inflow vorticity by the extension of the vortex tubes caused by the propeller induced flow [3].

To derive this result, consider the steady linearized Euler field equations in cylinderical coordinates for an imposed uniform flow U(r,θ),

$$U\frac{\partial}{\partial x}u_x + \frac{\partial U}{\partial r}u_r + \frac{\partial U}{\partial \theta}\frac{v_\theta}{r} = -\frac{1}{\rho}\frac{\partial p}{\partial x} \qquad (0.1)$$

$$U\frac{\partial}{\partial x}u_r = -\frac{1}{\rho}\frac{\partial p}{\partial r} \qquad (0.2)$$

$$U\frac{\partial}{\partial x}u_\theta = -\frac{1}{\rho r}\frac{\partial p}{\partial \theta} \qquad (0.3)$$

with the equation of continuity

$$\frac{\partial}{\partial x}(ru_x) + \frac{\partial}{\partial r}(ru_r) + \frac{\partial}{\partial \theta}(v_\theta) = 0 \qquad (0.4)$$

To obtain a field equation for p, differentiate (0.4) with respect to x and substitute (0.1) and (0.2). Eliminating the remaining terms containing velocity components using (0.2) and (0.3) yields

$$\nabla^2 p - \frac{1}{U}\frac{\partial U}{\partial r}\frac{\partial p}{\partial r} - \frac{1}{rU}\frac{\partial U}{\partial \theta}\frac{1}{r}\frac{\partial p}{\partial \theta} = 0 \tag{0.5}$$

To bring this equation to its final form, introduce the substitutions

$$V = 1/U \tag{0.6a}$$
$$P = p/U \tag{0.6b}$$

It follows that (0.5) will reduce to the following equation for the pressure

$$V\nabla^2 P - P\nabla^2 V = 0 \tag{0.7}$$

One method for solving this equation is to consider it as a Poisson's equation with the inhomogenous term depending linearly on the solution. Using a formal solution based on the Green's function leads to a Fredholm equation of the second kind

$$P(\vec{x}) = \frac{1}{4\pi}\iint_S \Delta p \frac{\partial}{\partial x}\frac{dS}{|\vec{x}-\vec{x}'|} - \frac{1}{2\pi}\iiint_V \frac{\frac{\nabla^2 V}{V} P}{|\vec{x}-\vec{x}'|} dV \tag{0.8}$$

Since an analytic expression for the Laplacian of the transformed velocity will not be generally available, this integral equation has to be solved numerically. Although numerical solution by direct or iteration methods does not pose intractable theoretical problems, it requires five-fold numerical integration over an infinite computational domain.

For this reason, it was decided to attempt a numerical solution of the above equation by turning the elliptic problem into a parabolic problem that could be solved by starting sufficiently far up stream of the propeller, and then marching the solution through the disc and sufficiently far down stream. Although it generates more complicated expressions to handle, this procedure side-steps many of the numerical difficulties inherent in solving (0.8)

1. Problem Formulation

We propose to solve the problem (0.7) on the computer by trying to split the operator in the operator equation (a PDE) into an axial second order differential equation with "constant coefficients" that has a simple and easily found Green's function [4]. The remaining radial and angular effects will be placed in the forcing function for this equation as feedback terms from the previous levels of solutions. In this way, five-fold integral evaluation can be avoided at the price of handling more complicated expressions and using functionals in the analysis rather than just the more classical functions.

Mathematically, this process of splitting the partial differential operator into two operators; an ordinary differential operator and an operator into a Banach space that contains the rest of the terms, can be justified provided that the 'lateral' derivatives are dominated by the axial derivatives. This assumption will be the one used to recast the problem without the five-fold integrals.

The ultimate output of this algorithm should be the harmonic content of the effective wake for selected harmonics. If we go through the computational process using a mesh in both the angular and radial directions, then the computational effort can be quite high to provide a sufficient density of calculated values to estimate the harmonic content. Another possible alternative is to recognize from the start that harmonic information is the desired output and write the algorithm to find the coefficients of a truncated Fourier series directly. Not only does this save on having to do the final FFT to find the coefficients from values on the mesh, but it also saves on the complexity of the mesh that must be used in the algorithm.

So, we will assume a Fourier expansion for both P and V, ie.

$$P(x,r,\theta) = \sum_{-\infty}^{\infty} p_m(x;r)e^{im\theta} \tag{1.1a}$$

$$(\nabla^2 V/V)(r,\theta) = \sum_{-\infty}^{\infty} \psi_m(r)e^{im\theta} \tag{1.1b}$$

and we will ignore those terms with an index with a magnitude that exceeds a fixed value, say M.

Using this representation, then (1.1) can be rewritten in the form

$$\left[\frac{\partial}{\partial x^2} - \left(\frac{m^2}{r^2} + V_0(r)\right)\right] p_m = \sum_{j=-\infty}^{\infty} V_{m-j} p_j - V_0 p_m - \frac{1}{r}\left(\frac{\partial}{\partial r} r \frac{\partial}{\partial r} p_m\right) \tag{1.2}$$

where the left hand side is the desired second order ordinary differential operator in x with 'constant coefficients' and the right hand side is a 'forcing function' that includes interactions between the various harmonics, and the radial distribution of the wake.

The ultimate advantage of this set of coupled ordinary differential equations would be if it allows quick and efficient solution of the problem at hand. That is the hypothesis that we intend to test. More immediate advantages to this form include

(1) the Green's function is easy to calculate

(2) the coupling between modes is only in the forcing term

(3) the mean radial velocity is already included

In these respects, the splitting of the PDE into an axial component and the rest has simplified the computational geometry, while making the functions to be used more complicated in form.

Since the Fourier components of the Laplacian of V divided by V can be determined once V is given, the system is a coupled set of linear equations for the Fourier coefficients of the transformed pressure. Given an initial of the solution, this system of equations can be used along with an iterative procedure to find the solution to the original problem.

In the next section, the Green's function for left hand side of equation (1.2) will be derived.

2. Deriving the Green's Function

Consider the prototype second order differential equation with constant coefficients and a general forcing term given by

$$\left[\frac{d^2}{dx^2} + f^2\right] p_m = q(x) \tag{2.1}$$

where f is a positive constant, and subject to the two sets of boundary conditions

$$p_m(-\infty, r) = 0 \tag{2.2a}$$
$$p_m(\infty, r) = 0 \tag{2.2b}$$

at the far upstream and downstream positions, and then on the two sides of the actuator disk

$$p_m(0^-; r) = -\tfrac{1}{2}\Delta p_m(r) \tag{2.3a}$$
$$p_m(0^+; r) = \tfrac{1}{2}\Delta p_m(r) \tag{2.3b}$$

that give the pressure jump associated with the disk. (It is assumed that the pressure jump is a known quantity that has also be resolved into its harmonic contents. It will be calculated from the usual Davidson Laboratory propeller programs like PLEXVAN [5], and PPEXACT. [6])

To solve this equation, use the method of variation of constants by assuming a solution of the particular form

$$p_m(x; r) = C_1(x; r)e^{fx} + C_2(x, r)e^{-fx} \tag{2.4}$$

subjected to the boundary conditions (2.3a) and (2.3b) at the actuator, and the additional boundary conditions

$$p_m(-X, r) = 0 \tag{2.5a}$$
$$p_m(X, r) = 0 \tag{2.5b}$$

where X is a specified large parameter.

These modified boundary conditions for far up and downstream have been used to insure a finite solution. The original conditions will be recovered by allowing the artifical parameter X to go to infinity.

Solving (2.1) using form (2.4) and applying the above boundary conditions yields a simple expression for the Green's function. Passing to the limit as X goes to infinity gives the Green's function

$$G_-(\xi, x) = e^{fx}\sinh(f\xi) \quad x < \xi < 0 \tag{2.6a}$$
$$= e^{f\xi}\sinh(fx) \quad \xi < x < 0 \tag{2.6b}$$

$$G_+(\xi, x) = e^{-fx}\sinh(f\xi) \quad 0 < x < \xi \tag{2.6c}$$
$$= e^{f\xi}\sinh(fx) \quad 0 < \xi < x \tag{2.6d}$$

that is associated with our problem.

In summary, the solution of (2.1) using the constructed Green's function is given by the expression

$$p_m(x;r) = -\tfrac{1}{2}\Delta p_m(r)e^{\kappa f} + \tfrac{1}{f}\int_{-\infty}^{0} G_-(\xi,x)\,q(\xi)\,d\xi \qquad x<0 \qquad (2.7a)$$

$$= \tfrac{1}{2}\Delta p_m(r)e^{-\kappa f} + \tfrac{1}{f}\int_{0}^{\infty} G_+(\xi,x)\,q(\xi)\,d\xi \qquad 0<x \qquad (2.7b)$$

In the next section, the full integral equation will be considered.

3. Numerical Approximation

Matching the terms in the simplified equation (2.1) with those in the full system (1.2) gives the identification

$$f_m(r) = \sqrt{\frac{m^2}{r^2} + k_0^2(r)} \qquad (3.1a)$$

$$q(x;r) = \sum_{j=-M}^{M} v_{m-j}\,p_j - v_0 p_m - \tfrac{1}{r}\left(\tfrac{\partial}{\partial r} r \tfrac{\partial}{\partial r} p_m\right) \qquad (3.1b)$$

Formally, the system (1.2) can be rewritten as the 'integral equation'

$$p_m(x;r) = -\tfrac{1}{2}\Delta p_m e^{\kappa x} + \tfrac{1}{f}\int_{-\infty}^{0} G_-(\xi,x)\,q(\xi;r,\rho)\,d\xi \qquad (3.1a)$$

$$= \tfrac{1}{2}\Delta p_m e^{-\kappa x} + \tfrac{1}{f}\int_{0}^{\infty} G_+(\xi,x)\,q(\xi;r,\rho)\,d\xi \qquad (3.1b)$$

where

$$q(x;r,\rho) = \sum_{j=-M}^{M} v_{m-j}\,p_j - v_0 p_m - \tfrac{1}{r}\left(\tfrac{\partial}{\partial r} r \tfrac{\partial}{\partial r} p_m\right) \qquad (3.2)$$

and the Green's function G is defined as (2.6) with the identifications in (3.1). Given the initial approximation, evaluate the forcing term, and then use equation (3.2) to find the next approximation to $P_m(x;r)$.

Since this iterative procedure is to be done numerically rather than by analytical means, some additional approximations are required. In particular, it is necessary to correctly approximate the behavior of the solution both far up and downstream of the actuator disc. In addition, the solution must be approximated using only a finite number of constants.

To estimate the behavior at infinity, the formal solution (2.7) indicates the first term decays at an exponential rate in the axial direction. The next section investigates the expected asymptotic behavior of the integral for a typical forcing function.

4. Asymptotic Behavior of the Forcing Function in the Tail

A propeller operating in a wake induces a pressure field, a component of which is independent of blade position or time. If this component is denoted by $P^{(0)}$, then the form of this zeroth order representation in cylinderical coordinates is given by

$$p^{(0)} = \frac{n}{4\pi^2 U}\sum_{\lambda=-\infty}^{\infty}\int_{r_h}^{r_0} s\,ds\int_{-\theta_b}^{\theta_b} d\theta'\,\Delta p_\lambda(s,\theta)\,e^{-i\lambda\theta'}A_\lambda(x,r,s)\,e^{i\lambda\theta} \qquad (4.1)$$

where A_λ is the propagation function. As $P^{(c)}(x,r,\theta)$ can be calculated by an existing program (ie. PPEXACT or PLEXVAN), then it is a natural starting point for the iterative procedure. From the nature of the Green's function, it appears that the decay rate in the tail that is typical for (4.1) will be the type of decay that expected in the solution.

The axial decay of $P^{(\sigma)}$ must be governed by the axial rate of decay of the propagation function. Turning to the formula for the propagation function, then

$$A_m(x,r,s) = \frac{\partial}{\partial \xi}\left[\frac{i}{\sqrt{rs}} Q_{|m|-\frac{1}{2}}\left(\frac{(x-\xi)^2+r^2+s^2}{2rs}\right)\right]\Big|_{\xi=0} \tag{4.2}$$

Taking the indicated derivatives, and simplifying by using the recursion relations for the Legendre functions of the second kind [7], then

$$A_m(x,r,s) = \frac{x}{rs\sqrt{rs}} \frac{|m|-\frac{1}{2}}{z^2-1}\left[Q_{|m|+\frac{1}{2}}(z) - zQ_{|m|-\frac{1}{2}}(z)\right] \tag{4.3}$$

where Z is given by

$$z = \frac{x^2+r^2+s^2}{2rs} \tag{4.4}$$

To estimate the rate of decay, recall the formula [8]

$$Q_{m+\frac{1}{2}}(z) = \frac{\Gamma(\frac{1}{2})}{2^{m+\frac{1}{2}}} \frac{\Gamma(m+\frac{1}{2})}{\Gamma(m+1)} \frac{1}{z^{m+\frac{1}{2}}} F\left(\frac{2m+1}{4}, \frac{2m+3}{4}, \frac{3}{2}, \frac{1}{z^2}\right) \quad |z^2|>1 \tag{4.5}$$

using hypergeometric functions. Doing an asymptotic expansion of the propagation function using the previous results shows that

$$A_m(x,r,s) \simeq \frac{|m|+\frac{1}{2}}{rs\sqrt{rs}} \frac{i}{|x|^{2m+2}}\left[C_{|m|-\frac{1}{2}} - \frac{C_{|m|+\frac{1}{2}}}{|x|^2}\right] \quad x>>1 \tag{4.6}$$

where the C's depend on m, but not on x. Naturally, the minimum rate of decay is associated with the case when $m = 0$.

Therefore, the dominant rate of decay for $P^{(c)}$, is x^{-1} where x is the distance in the axial direction. Therefore, the anticipated proper form of approximation is given by

$$q(\xi) = C_1|\xi|^{\alpha_1} \quad \xi \le -\bar{X} \tag{4.7a}$$

$$= \tilde{q}(\xi) \quad -\bar{X} < \xi < \bar{X} \tag{4.7b}$$

$$= C_2 \xi^{\alpha_2} \quad \bar{X} \le \xi \tag{4.7c}$$

where X, C_1, C_2, α_1, and α_2 are constants and $\tilde{q}(\xi)$ is a function. All these 'parameters' are to be determined from the data by an interpolation process.

This takes care of getting a reasonable approximation of the decay rate far from the actuator disc. In the region of the disc, a different type of approximation will be necessary.

5. Approximating the Forcing Function Near the Disc

In representing the forcing function near the actuator disc, it is necessary that the approximation (a) lead to an easy evaluation of the integrals involved in the integral equation, and (b) have a quick and easy method for fixing the parameters in the representation.

One simple option that satisfies both requirements is the use of low order polynomials between the points on a mesh. In the integral equation, this leads to a set of standard integrals; and determining the coefficients can be simplified by using the proper form for the polynomial.

One such choice is the use of cubic polynomials. Consider an arbitrary mesh on the closed interval [-X,X] and define a set of affine transformations from the closed interval [0,1] into the ith interval by

$$t = (\xi - \xi_i)/h_i \tag{5.1a}$$
$$d_i = (q_{i+1} - q_i)/h_i \tag{5.1b}$$
$$h_i = \xi_{i+1} - \xi_i \tag{5.1c}$$

where q_i and q_{i+1} are values of the function q(x) at the mesh points ξ_i and ξ_{i+1} respectively.

A particularly useful form of a cubic polynomial that interpolates at the two end points is given by

$$q_i(\xi) = t q_{i+1} + (1-t) q_i + h_i t(1-t)[(k_i - d_i)(1-t) - (k_{i+1} - d_i)t] \quad \xi_i \le \xi \le \xi_{i+1} \tag{5.2}$$

where the set of intermediate variables is defined as above. It follows automatically from the construction of the representation that it takes on the required values at the two end points. If the first derivative is taken with respect to the independent variable, and then evaluated at the two end points,

$$\frac{d}{d\xi} q_i(\xi_i) = k_i \tag{5.3a}$$
$$\frac{d}{d\xi} q_i(\xi_{i+1}) = k_{i+1} \tag{5.3b}$$

It follows immediately from these results that of the four coefficients necessary to define a cubic, two can be specified immediately given the functional values at the end points of the mesh. If the first derivatives of the function are known at the mesh points as well, then there is no need to solve any linear systems to fix the representation of the function on the interval. Furthermore, it follows from the definition of (5.2) that the representation is piecewise continuously differentiable in the independent variable, and locally has a continuous second derivative. Of course, by proper choice of the constants, one can have a function that is only piecewise continuous, or contains nothing higher than quadratic terms.

Another feature of this representation is that the global smoothness can be increased to the second derivative, if one does not insist on specifying the first derivative at the mesh points. To demonstrate this, calculate the second derivative of (2) and evaluate it at the mesh points, then

$$\frac{d^2}{d\xi^2} q_i(\xi)\Big|_{\xi_i} = \frac{2}{h_i}[3d_i - 2k_i - k_{i+1}] \tag{5.4a}$$

$$\frac{d^2}{d\xi^2}\phi_i(\xi)\Big|_{\xi_{i+1}} = \frac{1}{h_i}\left[3d_i - k_i - 2k_{i+1}\right] \qquad (5.4b)$$

The condition that the second derivative be continuous from one interval to the next can be rewritten as

$$h_{i+1}k_i + 2(h_{i+1}+h_i)k_{i+1} + h_i k_{i+2} = 3(h_{i+1}d_i + h_i d_{i+1}) \qquad (5.5)$$

If there are K mesh points in [-X,X] (or some smaller subset), then this provides (K-2) linear conditions on K unknowns. Furthermore, this portion of the system is tri-diagonal in general; which can be solved efficiently. To complete the linear system for the unknown coefficients, two more conditions must be imposed.

The representation just outlined for the interval [-X,X] is that for the spline approximation. It has the advantages of giving a global continuous second derivative on the interval with a minimal amount of computation. Of course, more information must be stored for each node in the mesh, the functional value and the approximate first derivative there. However, the mesh can be larger by using such an approximation and may ultimately save on the amount of computation necessary to have the desired accuracy on the solution. Secondly, spline approximations are not generally sensitive to the two conditions imposed to close the linear system.

Furthermore, by using a more detailed approximation at the initial stages of the coding, it is possible to fix the representation to be one of the simplier types without having to rewrite the code at all. Those coefficients that are fixed under another type of representation can be assigned the proper values and the numerical calculation go on as before.

6. Evaluation of the Integral

6.1. Far Field Contribution

Recall that in section 4, the approximate form of the forcing function was given by

$$\phi(\xi) = c_1|\xi|^{-\alpha_1} \qquad \xi \le -X \qquad (6.1.1a)$$

$$= \phi_i(\xi) \qquad \xi_i \le \xi \le \xi_{i+1} \qquad (6.1.1b)$$

$$= c_2 \xi^{-\alpha_2} \qquad X \le \xi \qquad (6.1.1c)$$

where $\xi_1 = -X$, $\xi_N = X$, and on the mesh

$$\phi_i(\xi) = t\phi_{i+1} + (1-t)\phi_i + h_i t(1-t)\left[(\kappa_i - d_i)(1-t) - (\kappa_{i+1} - d_i)t\right] \qquad (6.1.2)$$

where the constants are defined as in (5.1a)-(5.1c); and the various remaining constants are assumed to be given. To summarize, this is an approximation that has globally (outside the origin) continuous second derivatives, and decays like a multi-pole in the far field. This will be our general class for the forcing function.

Since the spline approximation above is composed of elementary functions, and so is the Green's function for this problem, then it is possible to evaluate the integral numerically as a collection of subterms that can be calculated by the computer. This will be more efficient and accurate than using a numerical quadrature scheme, and allows the use of a non-uniform mesh.

Let us consider the case of the far field contribution first. Define a set of standard functions by

$$E_\alpha(x) \equiv \int_x^\infty e^{-t}/t^\alpha \, dt \qquad x > 0 \qquad (6.1.3a)$$

$$S_\alpha(x,a) \equiv \int_1^{1+x} \sinh(at)/t^\alpha \, dt \qquad (6.1.3b)$$

where α is an integer constant and x is a parameter. These functions will be used to replace the results of integrating the Green's function with the tail in order to have the far field contribution. Define FT1 and FT2 by

$$FT1(x,\alpha,f) = \int_x^\infty e^{-f\xi}/\xi^\alpha \, d\xi = f^{\alpha-1} E_\alpha(fx) \qquad (6.1.3c)$$

$$FT2(x,\alpha,f) = \int_X^x \sinh(f\xi)/\xi^\alpha \, d\xi = \frac{1}{X^{\alpha-1}} S_\alpha\left(\frac{x-X}{X}, fX\right) \qquad (6.1.3d)$$

then it follows from (2.6) that

$$\int_{-\infty}^{-X} G_-(\xi,x)(-\xi)^{-\alpha} d\xi = FT1(-x,\alpha,f)\sinh(fx) - FT2(-x,\alpha,f)e^{fx} \qquad x < -X \qquad (6.1.4a)$$

$$= FT1(X,\alpha,f)\sinh(fx) \qquad -X < x < 0 \qquad (6.1.4b)$$

$$\int_X^\infty G_+(\xi,x)(\xi)^{-\alpha} d\xi = FT1(X,\alpha,f)\sinh(fx) \qquad 0 < x < X \qquad (6.1.4c)$$

$$= FT1(x;\alpha,f)\sinh(fx) + FT2(x;\alpha,f)e^{fx} \qquad X < x \qquad (6.1.4d)$$

The far field contribution can be calculated if the functions in (6.1.4a)-(6.1.4b) can be evaluated.

A number of different approximation schemes were studied for these functions. Since accuracy was very important, the best method required reducing the calculation to the special case when the exponent in the denominator is one (an exponential integral). This is possible by doing integration by parts since the intermediate terms are always elementary functions that can evaluated quite simply.

Once the terms resulting from process of integrating by parts have been evaluated and summed, then special case when the exponent is one can be evaluated by a rational function [9].

The case of (6.1.4b) did not lend itself to either rational function approximation or a series expansion to sufficient accuracy. Instead, the integral is approximated by using Simpson's rule with only two intervals. The low number of intervals was felt justified since the parameter $\frac{s-\beta}{\gamma}$ is rarely more than a few percent. On this interval, a parabolic approximation should be more than enough since the integrand will not vary greatly.

Therefore, both functions are evaluated using recursion based on integration by parts, followed by another procedure to calculate the special case.

6.2. Computation of the Far Field Contribution

For the sake of clarity, the recursion relationships below are reproduced in a 'structured' algorithmic language to make them more understandable. The first recursion relation will be for the pure exponential kernel.

```
x   <--  Passed
f   <--  Passed
ia  <--  Passed

F   <--  0
C   <--  1

DO UNTIL ia = 1
     F   <--  F + (C/ia-1)*exp(-f*x)/(x**(ia-1))
     C   <--  C * (-1*f)/(ia-1)
     ia  <--  ia - 1
ENDDO;
```

The second recursion relation is just a little more difficult to represent because the kernel changes from the hyperbolic sine to the hyperbolic cosine as the repeated integration by parts is performed. This means that separate code is required for each case and a flag must be used to indicate which case is being evaluated.

```
fX  <--  Passed
ia  <--  Passed
d   <--  Passed
```

```
C     <--  1
FLAG <--  S

DO UNTIL ia=1
     IF FLAG=S
        THEN DO
                FU   <-- sinh((1+d)*fX)/(1+d)**(ia-1)
                FL   <-- sinh(fX)
                FLAG<-- C
             ENDDO

         ELSE DO
                FU   <-- cosh((1+d)*fX)/(1+d)**(ia-1)
                FL   <-- cosh(fX)
                FLAG<-- S
             ENDDO

      ENDIF
      DO
           F    <--  F + (C/ia-1)*(FL-FU)
           C    <--  C * (fX/ia-1)
      ENDDO
 ENDDO
```

These special approximations mentioned above are used to calculate the special cases when the parameter α is 1 and the values of F and C are used to calculate the final approximation to the values of the special integrals. These are then used in the calculation of the far field contribution to the integral, and were tested to be quite accurate (to within a few percent) of the values determined by other methods. Sample calculations are given in the table below.

	Sinhi			
ia	d	f	Computed	Value
2	.01	4	0.27566	0.275659
2	.1	4	3.0315	3.03146
4	.01	4	0.27292	0.272921
4	.1	4	2.7469	2.74671

	Ei			
ia	x	f	Computed	Value
2	1.	1.	.14849	.14850
2	1.	1.25	.10350	.10348
2	1.	1.5	.073101	.0731008
2	1.	2.	.037529	.0375343
4	2.	1.	.025020	.0250228

The agreement between the recursion relationship and the value obtained by other methods is quite good. This lead to putting the subroutines in the code as FORTRAN versions of the algorithms given above. Since the parameter α may not determined at the outset of the calculation, then there will be repeated computation of the far field contribution in the

then there will be repeated computation of the far field contribution in the inner loops of the calculation. As a result, the approximations of the special cases $\alpha = 1$ were chosen to be as quick computationally as accuracy would permit.

(This concludes the immediate discussion about the accuracy of the far field contribution and how to compute it efficiently.)

6.3. Near Field Contribution

On the mesh, the forcing function has been approximated by a set of cubic splines. The contribution of the forcing function on the mesh is therefore approximated by

$$\int_c^X G_\alpha(\xi,x)\phi(\xi)\,d\xi \simeq \sum_{i=1}^{N-1} \int_{\xi_i}^{\xi_{i+1}} G_\alpha(\xi,x)\phi_i(\xi)\,d\xi \qquad (6.3.1)$$

where it is understood that

$$\xi_1 = 0^+ \qquad (6.3.2a)$$
$$\xi_N = X \qquad (6.3.2b)$$

(when x is negative, the calculation proceeds in the same way using the symmetry relations discussed in section 6.5).

Making the linear transformation

$$t = (\xi - \xi_i)/h_i \qquad (6.3.3a)$$
$$h_i = \xi_{i+1} - \xi_i \qquad (6.3.3b)$$

of the integration variable, then it follows that the integral can be rewritten as a linear combination of a class of subintegrals, ie.

$$\int_c^X G_\alpha(\xi,x)\phi(\xi)\,d\xi = \sum_{i=1}^{N-1} h_i \Big\{ \phi_{i+1}\int_0^1 G_\alpha(\xi_i+h_i t,x)\,t\,dt + \phi_i \int_c^1 G_\alpha(\xi_i+h_i t,x)(1-t)\,dt + h_i(x-\xi_i)\int_c^1 G_\alpha(\xi_i+h_i t,x)\,t(1-t)\,dt - h_i(x_{i+1}-\xi_i)\int_0^1 G_\alpha(\xi_i+h_i t,x)\,t^2(1-t)\,dt \Big\} \qquad (6.3.4)$$

Tabulation of these four subintegrals will lead to evaluation of the near field integral by just multiplying by the appropriate constants in the spline approximation and adding the various terms together.

To see how the calculation is done in more detail, consider the case of the first subintegral. By construction, the value of x will come from the set of mesh points. Therefore, there are only two possibilities to consider, namely

$$x \le \xi_i \qquad (6.3.5a)$$
$$\xi_{i+1} \le x \qquad (6.3.5b)$$

given that the value of i has been fixed for the moment. In the case of (6.3.5a), it follows from the definition of the Green's function (2.6) that

$$\int_c^1 G_\alpha(\xi_i+h_i t,x)\,t\,dt = \sinh(\alpha x)\int_c^1 t\exp\{-\alpha(\xi_i+h_i t)\}\,dt \qquad (6.3.6a)$$

while for (5b), the resulting expression is

$$\int_c^1 G_\alpha(\xi_i+h_i t,x)\,t\,dt = e^{-\alpha x}\int_c^1 t\sinh\{\alpha(\xi_i+h_i t)\}\,dt \qquad (6.3.6b)$$

To simplify the notation, let us introduce two special functions defined by

$$E_{NM}(a,b) \equiv \int_c^1 t^N (1-t)^M \exp-(a+bt)\,dt \qquad (6.3.7a)$$

$$S_{NM}(a,b) \equiv \int_c^1 t^N (1-t)^M \sinh(a+bt)\,dt \qquad (6.3.7b)$$

Then, it follows by definition that

$$\int_c^1 G_+(\xi_i+h_i,t,x)\,t\,dt = E_{1c}(f\xi_i, fh_i)\sinh(fx) \qquad x \le \xi_i \qquad (6.3.8a)$$

$$= S_{1c}(f\xi_i, fh_i)\,e^{-fx} \qquad \xi_{i+1} \le x \qquad (6.3.8b)$$

In the same fashion, it follows immediately from the definitions that

$$\int_c^1 G_+(\xi_i+h_i,t,x)(1-t)\,dt = E_{c1}(f\xi_i, fh_i)\sinh(fx) \qquad x \le \xi_i \qquad (6.3.9a)$$

$$= S_{c1}(f\xi_i, fh_i)\,e^{fx} \qquad \xi_{i+1} \le x \qquad (6.3.9b)$$

$$\int_c^1 G_+(\xi_i+h_i,t,x)\,t(1-t)^2\,dt = E_{12}(f\xi_i, fh_i)\sinh(fx) \qquad x \le \xi_i \qquad (6.3.9c)$$

$$= S_{12}(f\xi_i, fh_i)\,e^{-fx} \qquad \xi_{i+1} \le x \qquad (6.3.9d)$$

$$\int_c^1 G_+(\xi_i+h_i,t,x)\,t^2(1-t)\,dt = E_{21}(f\xi_i, fh_i)\sinh(fx) \qquad x \le \xi_i \qquad (6.3.9e)$$

$$= S_{21}(f\xi_i, fh_i)\,e^{-fx} \qquad \xi_{i+1} \le x \qquad (6.3.9f)$$

Tabulation of the functions given by (6.3.9a) and (6.3.9b) would only need to be done once at the beginning of the problem, provided that the mesh did not have to be refined or moved during the calculation. Evaluating the contribution from the near field becomes a process of looking up the constants in the spline approximation and forming the indicated linear combinations. Formulas (6.3.8a)-(6.3.8b) and (6.3.9a)-(6.3.9f) are particularly simple to program, making for a computationally efficient scheme.

The tabulation of the functions in (6.3.7a) and (6.3.7b) can be done using exact expressions if the evaluation is done by using recurrence relations. As an example, consider the case of (6.3.7a) and define an auxiliary set of special functions by

$$C_{NM}(a,b) \equiv \int_0^1 t^N (1-t)^M \cosh(a+bt)\,dt \qquad (6.3.10)$$

Integrating (6.3.7a) by parts yields the expression

$$S_{NM}(a,b) = \frac{1}{b}\left\{ \left[t^N(1-t)^M \cosh(a+bt)\right]\Big|_c^1 + MC_{N,M-1}(a,b) - NS_{N-1,M}(a,b) \right\} \qquad (6.3.11a)$$

and likewise,

$$C_{NM}(a,b) = \frac{1}{b}\left\{ \left[t^N(1-t)^M \sinh(a+bt)\right]\Big|_0^1 + MS_{N,M-1}(a,b) - NC_{N-1,M}(a,b) \right\} \qquad (6.3.11b)$$

It follows immediately from (6.3.11a) and (6.3.11b) that (6.3.7a) can be calculated by recursion given the values of (6.3.7a) and (6.3.10) when both N and M are zero. However, those cases are just standard elementary integrals. Therefore, the entire tabulation procedure can be done by starting with a closed expression for the special functions and extending the results by recursion on the indices.

Integrating (6.3.7b) by parts leads to the recursion relationship

$$E_{NM}(a,b) = \frac{1}{b}\left\{\left[t^N(1-t)^M e^{-(a+bt)}\right]\Big|_0^1 + M E_{N,M-1}(a,b) - N E_{N-1,M}(a,b)\right\} \qquad (6.3.12)$$

which is even simplier since no second set of special functions is needed to close the recursion.

6.4. Computation of the Near Field Contribution

The evaluation of the basic subintegrals can be simplified greatly at the coding stage. Note the ascending sequence of function evaluations that is involved.

```
C00(a,b) = (sinh(a+b)-sinh(a))/b                          (6.4.2a)

S10(a,b) = (cosh(a+b)-C00(a,b))/b                         (6.4.2b)
S01(a,b) = (C00(a,b)-cosh(a))/b                           (6.4.2c)

C20(a,b) = (sinh(a+b)-2*S10(a,b)))/b                      (6.4.2d)
C11(a,b) = (S10(a,b)-S01(a,b))/b                          (6.4.2e)
C02(a,b) = (2*S01(a,b)-sinh(a))/b                         (6.4.2f)

S21(a,b) = (C20(a,b)-2*C11(a,b))/b                        (6.4.2g)
S13(a,b) = (2*C11(a,b)-C02(a,b))/b                        (6.4.2h)
```

where the parameters are

$$a = \xi_i f_m(r) \qquad (6.4.3a)$$

$$b = h_i f_m(r) \qquad (6.4.3b)$$

Doing the same thing with the other part of the kernel, it follows from the recursion relation above that

```
E00(a,b) = (exp(-a)-exp-(a+b))/b                          (6.4.4a)

E10(a,b) = (E00(a,b)-exp-(a+b))/b                         (6.4.4b)
E01(a,b) = (exp(a)-E00(a,b))/b                            (6.4.4c)

E20(a,b) = (2*E01(a,b)-exp-(a+b))/b                       (6.4.4d)
E11(a,b) = (E01(a,b)-E10(a,b))/b                          (6.4.4e)
E02(a,b) = (exp(a)-2*E01(a,b))/b                          (6.4.4f)

E21(a,b) = (2*E11(a,b)-E20(a,b))/b                        (6.4.4g)
E12(a,b) = (E02(a,b)-2*E11(a,b))/b                        (6.4.4h)
```

where the parameters are the same as above.

If you study the set of recursion relationships given above, it should become clear that they are essentially the same. This is a result of the fact that the kernel is basically exponential in behavior. As a result, only a single set of recursion relationships should be necessary in the code.

To simplify the appearance of the code, the actual recursion relationships were encoded as statement functions. Not only does this make the code more compact to the reader, but it also makes it possible the modify the recursion relationship in a simple fashion if necessary.

The code was written to reduce CPU time where possible. For one thing, the evaluation of the near field contributions can be computed before the actual iteration code and stored in an array. Two, the subroutine that evaluates the integral involving the Green's function uses the pre-digested results in the inner-most loop.

Third, subroutine overhead was reduced by performing the recursion calculations directly in the subroutine rather than breaking it out as a separate subroutine. There is one section of code that does the actual recursion calculation, and it is reached by performing a program jump. Control is returned to the proper section of the program by using a computed GOTO with a flag to the point of return.

These modifications do increase somewhat the usage of memory and make the code less 'structured'. However, they reduced the amount of computation that is being done in the inner loops of the algorithm and reduced the amount of overhead involved with subroutine calls. So, execution speed of the program is increased.

6.5. Application of Symmetry

If we use the definition of the Green's function for this problem as given in (2.6), then it follows by definition that the Green's function satisfies the symmetry relationship

$$G_-(\xi,x) = -G_+(-\xi,-x) \tag{6.5.1}$$

The question that will be answered in this section is how this symmetry relationship can be used to simplify the calculation of the near field contribution.

Consider the case of the first subintegral on the ith interval (where i is less than N/2), then by definition of the spline approximation,

$$h_i \int_{\xi_i}^{\xi_{i+1}} G_-(\xi,x)\,t\,dt = \int_{\xi_i}^{\xi_{i+1}} G_-(\xi,x)\,\frac{\xi-\xi_i}{h_i}\,d\xi \tag{6.5.2}$$

If the mesh points have been chosen to be symmetric about the origin, then it follows from the construction of the mesh that

$$\xi_i = -\xi_{N-i} \tag{6.5.3}$$

As an immediate result, it follows from (5.1c) that

$$h_i = \xi_{i+1} - \xi_i = -\xi_{N-i-1} - (-\xi_{N-i}) = \xi_{N-i} - \xi_{N-i-1} = h_{N-i-1} \tag{6.5.4}$$

Substituting these identities into (6.5.2), then it follows that

$$h_i \int_{\xi_i}^{\xi_{i+1}} G_-(\xi,x)\,t\,dt = -\int_{\xi_{N-i-1}}^{\xi_{N-i}} G_-(-\xi',-x')\,\frac{\xi_{N-i}-\xi'}{h_{N-i-1}}\,d\xi' \tag{6.5.5}$$

after making the change of variables,

$$\xi = -\xi' \tag{6.5.6a}$$

$$x = -x' \tag{6.5.6b}$$

Using (6.5.1) to change the domain of the kernel function, it follows from (6.5.5) that

$$h_i \int_{\xi_i}^{\xi_{i+1}} G_-(\xi, x) f dt = -\int_{\xi_{N-i}}^{\xi_{N-i+1}} G_+(\xi', x') \frac{\xi' - \xi_{N-i+1}}{\xi_{N-i} - \xi_{N-i+1}} d\xi' \tag{6.5.7}$$

In the same fashion, if the second subintegral were chosen, the same substitutions demonstrates the relationship

$$h_i \int_{\xi_i}^{\xi_{i+1}} G_-(\xi, x)(1-\tau) dt = -\int_{\xi_{N-i-1}}^{\xi_{N-i}} G_+(\xi', x') \frac{\xi' - \xi_{N-i}}{\xi_{N-i-1} - \xi_{N-i}} d\xi' \tag{6.5.8}$$

Similar results hold for the remaining two subintegrals.

The remaining step is to see how the symmetry relations above are reflected in the special functions defined in (6.3.7a)-(6.3.7b). If the definition of the Green's function and the special functions are combined with the symmetry relations above, it follows that

$$S_{MM}(f\xi_i, fh_i) = -S_{NN}(f\xi_{N-i-1}, fh_{N-i-1}) \tag{6.5.9a}$$

$$E_{MM}(f\xi_i, fh_i) = E_{NN}(f\xi_{N-i-1}, fh_{N-i-1}) \tag{6.5.9b}$$

Therefore, by invoking the symmetry relationships, the calculations of the basic integrals can be reduced by about one-half over what a straight calculation would require.

7. The Recursion Relation for the k's

One feature about the problem at hand is that the solution to the integral equation should be one that has continuous second derivatives (almost everywhere), but can not be globally continuous. After all, there is a clear jump in the solution from passing from one side of the actuator disk to the other. Therefore, it is proper for a strong solution of this problem to be piecewise continuous in the second derivative, with the only discontinuities being at the origin.

Rather than having one set of splines cover the entire interval there will be one set on the submesh [-X,0] and another set on the submesh [0,X]. The requirement that the spline approximation have globally continuous second derivatives on each half of the mesh leads to the set of linear equations (5.5) for the k's. However, this system is underdetermined and requires two additional conditions on each half of the mesh. The natural places to specify these conditions are at the boundaries of the submesh.

Since the flow going into the actuator disk is actually something that we are trying to determine, the only conditions that can be applied across the actuator disk are those for conservation of the basic fluid properties; mass, momentum, and energy. Since the fluid is assumed incompressible, then the energy equation decouples and gives no useful information. On the other hand, the conditions for conservation of mass and momentum have already been applied to give the equations of motion, and the resulting integral equation (3.2).

That leaves just the far endpoints for the specification of the additional conditions on the coefficients. By the construction of the spline, the requirement that the spline have the same value as the forcing function at the endpoint is automatically satisfied; then the natural conditions to specify are the values of the first and second derivative. This requirement would make the approximation for the forcing function have global continuous second derivatives on each of the two half-lines.

Given the values of the first and second derivatives at -X, and X, the algebraic condition for global continuity of the second derivative (excluding the origin) reduces to

$$h_i K_{i+2} = 3(h_{i+1} d_i + h_i d_{i+1}) - 2(h_i + h_{i+1}) k_{i+1} - h_{i+1} k_i \qquad (7.1)$$

which is just a recursion relationship for the remaining coefficients in the representation.

The remaining task is to specify the first and second derivatives at those places where the mesh ends. An explicit specification will be avoided at this time by having the values of the first and second derivatives specified.

It begs the question by leaving the issue open about how the parameters in the far-field approximation are to be fixed. To try and write them in terms of values on the interior of the mesh would just create a circular loop. However, it is possible using the information at hand to calculate the functional value at the far mesh point, and at one or more

points outside the mesh in the far field. The procedure for estimating the rate of decay given these values will be presented later.

The next issue to be studied here will be the stability of the recursion formula derived above.

7.1. Stability Analysis of the Recursion Relationship

Although having a recursion relation to evaluate reduces computation time as compared to solving a tri-diagonal system; there is the potential problem of a numerical instability arising. Looking the recursion relation (7.1), it is to be expected that errors will grow from one stage to the next rather than decay.

To prove this conjecture, consider how errors are propagated through the calculation. The recursion relation for the error when the grid is uniform is

$$\epsilon_{i+2} = -4\epsilon_{i+1} + \epsilon_i \qquad (7.1.1)$$

where ϵ_i is the absolute error in the ith coefficient k. To solve this difference relation, make the transformation

$$\epsilon_i = \lambda^i \qquad (7.1.2)$$

which leads the to quadratic equation

$$\lambda^2 + 4\lambda + 1 = 0 \qquad (7.1.3)$$

which has the two real solutions,

$$\lambda_1 = -2 - \sqrt{3} \simeq -3.732051 \qquad (7.1.4a)$$

$$\lambda_2 = -2 + \sqrt{3} \simeq 0.267949 \qquad (7.1.4b)$$

Clearly, the first root will lead to an oscillating numerical instability in the calculation of the k's.

With the growth rate in the error established, it is time to consider how the growth is triggered. The recurrence relationship for the error in the k's (including round-off) can be written as

$$\epsilon_{i+2} + 4\epsilon_{i+1} + \epsilon_i = 3(\delta_i + \delta_{i+1}) \qquad (7.1.5)$$

using the notation above, and δ_i and δ_{i+1} are the round-off errors in the calculation of product. To study the absolute error growth with index, use the variation of parameters method using a trial solution of the specific form

$$\epsilon_i = C_1(i)\lambda_1^i + C_2(i)\lambda_2^i \qquad (7.1.6)$$

where λ_1 and λ_2 are the roots of the homogenous quadratic equation given above (7.1.4). Since (7.1.6) has two unknown functions of the index to be solved for, and there is only one equation, it follows by the method of variation of parameters that a second condition should be imposed on the

equation. To simplify matters, the second condition will be taken as

$$[C_1(i+1) - C_1(i)]\lambda_1^i + [C_2(i+1) - C_2(i)]\lambda_2^i = 0 \qquad (7.1.7a)$$

Therefore, putting (7.1.6) into (7.1.7), and using that and λ_1 and λ_2 are the roots of the quadratic equation in question, then the system becomes

$$4[C_1(i+2) - C_1(i+1)]\lambda_1^{i+1} + [C_1(i+2) - C_1(i)]\lambda_1^i + 4[C_2(i+2) - C_2(i+1)]\lambda_2^{i+1} + \qquad (7.1.7b)$$

$$[C_2(i+1) - C_2(i)]\lambda_2^i = -3(\delta_i + \delta_{i+1})$$

to solve for the two known coefficient functions. Solving (7.1.7a) for C1 in terms of C2 simplifies (7.1.7b) to the single first order difference equation

$$C_2(i+2) - C_2(i+1) = -\frac{3(\delta_i + \delta_{i+1})}{\lambda_2^{i+1}(1 - \lambda_2/\lambda_1)} \qquad (7.1.8)$$

This simple difference equation has the solution

$$C_2(i) = \frac{-3}{1 - \lambda_2/\lambda_1}\sum_{j=1}^{i-1}\frac{\delta_{j-1} + \delta_j}{\lambda_2^{j+1}} \qquad (7.1.9)$$

and substituting this back into (7.1.7b) and solving for the other unknown function gives

$$C_1(i) = \frac{3\lambda_2}{1 - \lambda_2/\lambda_1}\sum_{j=1}^{i-1}\frac{\delta_{j-1} + \delta_j}{\lambda_1^{j+1}} \qquad (7.1.9b)$$

Returning to the original equation for the error, then the general solution has the form

$$\epsilon_i = C_1\lambda_1^i + C_2\lambda_2^i + \frac{3}{1 - \lambda_2/\lambda_1}\sum_{j=1}^{i-1}\left[\frac{\lambda_2}{\lambda_1}\lambda_1^{i+1-j} - \lambda_2^{i+1-j}\right](\delta_{j-1} + \delta_j) \qquad (7.1.10)$$

where C1 and C2 are now constants to be specified when the initial errors are given.

As (7.1.10) clearly shows, even in the case where there is no initial error, the round-off error will excite a second mode of growth that is almost that of the unstable root. The largest contribution will clearly be due to the initial truncation errors near the ends of the mesh. So, if the error is controlled at this stage in the calculation, difficulties may be avoided later.

This suggests that the d values should be computed in double precision to reduce the round off error that is introduced at that stage of the calculation. That will help delay the onset of significant error growth, but can not completely eliminate it.

8. Testing the Recursion-Based Program

Once the bulk of the code was in place, it was necessary to test the general purpose integration routine to verify its accuracy on the contributions from both the near and far field. This was done by using three tests

(1) Cubic Polynomials

(2) A Quartic Polynomial

(3) Tail Test

Since the representation of cubic polynomials by splines is exact, the choice of a cubic for the forcing function would lead to a known result that would show if there were any logical flaws in the main code. However, the case of a general cubic leads to a divergent integral.

Therefore, the test case was to use the product of a cubic polynomial with a function that was one on the mesh, and zero outside. This leads to a simple set of integrals that will be presented a little later in this discussion.

Once the code gives the correct answers in the case of a modified cubic polynomial on the mesh, then it is necessary to assess the error caused by only using the cubic polynomial representation. As an indication of the truncation error, the forcing function was taken to be the quartic (x^4) on the mesh, and zero outside. Again, this particularly simple form for the forcing function leads to exact solutions that can be used to judge the truncation error that is induced in the procedure by using cubic splines rather than the exact forcing function representation.

The final test was to see how well the numerical procedure did about the modelling of a pure decay, the tail test.

8.1. The Cubic Test

As mentioned before, the use of a forcing function which is a cubic on the mesh leads to a test case that will demonstrate the logical errors in both the spline approximation algorithm as well as the algorithm to approximate the integral of the forcing function and the Green's function for this problem.

The first order of business is to construct the results of the exact integration using a general forcing function that vanishes outside the mesh. (Attention will be restricted to the case where the field point has a positive x coordinate since the other case will follow by a symmetry transformation.) There are two cases, when the field point is one of the mesh points, and when it lies outside the mesh points.

In the first case, the integral must be split into three domains of integration, namely

$$\int_0^\infty G_+(\xi,x)q(\xi)\,d\xi = \int_0^x G_+(\xi,x)q(\xi)\,d\xi + \int_x^X G_+(\xi,x)q(\xi)\,d\xi + \int_X^\infty G_+(\xi,x)q(\xi)\,d\xi \qquad (8.1.1)$$

The last integral will vanish by definition of the class of the test forcing functions, while the remaining ones become

$$\int_0^\infty G_+(\xi,x)q(\xi)\,d\xi = e^{-fx}\int_0^x q(\xi)\sinh(f\xi)\,d\xi + \sinh(fx)\int_x^X q(\xi)e^{-f\xi}\,d\xi \qquad (8.1.2)$$

by the virtue of the definition of the Green's function. Likewise, the

integral in the case where the field point is outside the mesh simplifies to

$$\int_c^{\infty} G_+(\xi,x)\,q(\xi)\,d\xi = e^{-fx}\int_c^{X} q(\xi)\sinh(f\xi)\,d\xi \qquad (8.1.3)$$

by the definition of the class of test forcing functions and the Green's function.

For the case where the test forcing function is cubic on the mesh, it is sufficient to verify the algorithms for the special subcases

(1) $q(\xi) = 1 \qquad 0 < \xi < X$

(2) $q(\xi) = \xi \qquad 0 < \xi < X$

(3) $q(\xi) = \xi^2 \qquad 0 < \xi < X$

(4) $q(\xi) = \xi^3 \qquad 0 < \xi < X$

for then the general result can be established by creating the proper linear combinations of these cases.

Straight forward integration shows that

Case 1

$$\int_0^{\infty} G_+(\xi,x)\,q(\xi)\,d\xi = \frac{1}{f}\left[\cosh(fx)-1\right]e^{-fx} + \frac{1}{f}e^{-f\xi}\Big|_x^X \sinh(fx) \qquad c<x<X \qquad (8.1.4a)$$

$$= \frac{1}{f}\left[\cosh(fX)-1\right]e^{-fx} \qquad X<x \qquad (8.1.4b)$$

Case 2

$$\int_c^{\infty} G_+(\xi,x)\,q(\xi)\,d\xi = \frac{1}{f^2}\left[(fx)\cosh(fx)-\sinh(fx)\right]e^{-fx} + \frac{1}{f^2}\left\{(f\xi+1)e^{-f\xi}\right\}\Big|_c^X \sinh(fx) \qquad c<x<X \qquad (8.1.4c)$$

$$= \frac{1}{f^2}\left[(fX)\cosh(fX)-\sinh(fX)\right]e^{-fx} \qquad X<x \qquad (8.1.4d)$$

Case 3

$$\int_0^{\infty} G_+(\xi,x)\,q(\xi)\,d\xi = \frac{1}{f^3}\left\{\left[(fx)^2+2\right]\cosh(fx)-2(fx)\sinh(fx)-2\right\}e^{-fx} + \frac{1}{f^3}\left\{\left[(f\xi)^2+2(f\xi)+2\right]e^{-f\xi}\right\}\Big|_c^X \sinh(fx) \qquad c<x<X \qquad (8.1.4e)$$

$$= \frac{1}{f^3}\left\{\left[(fX)^2+2\right]\cosh(fX)-2(fX)\sinh(fX)-2\right\}e^{-fx} \qquad X<x \qquad (8.1.4f)$$

Case 4

$$\int_0^{\infty} G_+(\xi,x)\,q(\xi)\,d\xi = \frac{1}{f^4}\left\{\left[(fx)^3+6(fx)\right]\cosh(fx)-3\left[(fx)^2+2\right]\sinh(fx)\right\}e^{-fx} + \frac{1}{f^4}\left\{\left[(f\xi)^3+3(f\xi)^2+6(f\xi)+6\right]\right\} \qquad (8.1.4g)$$

(8.1.4h)

which were coded into a FORTRAN program to calculate the exact results and display them.

The main program code was altered to include a set of test subroutines that could be invoked interactively. One such choice was to choose the cubic test option where the coefficients of the polynomial could be given from the terminal. The calculation was done, and the results were printed out. The computed answers could then be compared with the answers given by the exact code for the same mesh points.

For the cubic polynomial tests, 20 intervals were used with the maximum extent of the mesh being 16. (This implies that h = 2.0 uniformly on the mesh.) The exact first and second derivative at the far end point of the submesh was also supplied. The results using the test cubics are compared with the results obtained by a separate program that did the calculation using a polynomial forcing function exactly.

	Constant		Linear		Quadratic	
X	Test	Exact	Test	Exact	Test	Exact
0.0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2.0	0.86466	0.86466	2.00000	2.00000	5.2795	5.7292
4.0	0.98168	0.98168	3.9999	4.0000	17.962	17.963
6.0	0.99750	0.99750	5.9996	5.9996	379.88	379.89
8.0	0.99950	0.99950	7.9971	7.9972	659.51	659.51
10.0	0.99872	0.99872	9.9789	9.9789	1016.4	1016.4
12.0	0.99084	0.99084	11.844	11.844	1433.4	1433.4
14.0	0.93233	0.93233	12.850	12.850	1783.8	1783.8
16.0	0.50000	0.50000	7.500	7.500	1130.0	1130.0

	Cubic	
X	Test	Exact
0.0	0.00000	0.000000
2.0	19.995	20.000
4.0	87.985	87.985
6.0	251.89	251.89
8.0	559.17	559.17
10.0	1053.8	1053.9
12.0	1754.5	1754.5
14.0	2492.0	2492.0
16.0	1709.0	1709.0

The proceeding table of results indicates that the calculated answer is very close to the exact answer calculated by another method. In most cases, there is no difference. Where a difference exists, it is probably due to round-off. This suggests that the algorithm is capable of reproducing the exact results when a cubic polynomial is taken as being the forcing function on the mesh. In particular, this shows that no significant logical or typing errors existed in the code that did the evaluation of the integral of the Green's function for the mesh. In the next section, an estimate of the truncation error will be made for this method.

8.2. The Quartic Polynomial Test

The use of the quartic polynomial for testing was to determine the truncation errors associated with this algorithm. The quartic polynomial was chosen because it does not have an exact representation in terms of cubic polynomials, and yet the exact solutions were easy to compute. It should be understood that the quartic polynomial is also considered to be multiplied by a function which is one on the mesh, and zero outside the mesh in order to avoid evaluation of a divergent intergral.

8.2.1. Truncation Error Test 1

The first test of truncation error used a forcing function that was quartic on the mesh using a cubic Hermite interpolation. The use of this approximation to fix the k's for the quartic was to avoid error due to approximation of the first derivative of the function at the mesh points. The resulting error is solely produced by the approximation of the fourth order polynomial by a cubic polynomial.

The results of the test calculation are summarized in the table below along with the results of the second truncation test.

8.2.2. Truncation Error Test 2

The second test of truncation error used the same quartic forcing function on the mesh as mentioned in the previous section. However, rather than use the exact value of the first derivative at the mesh points, the full spline approximation was used where the first derivative is approximated from the given functional values. The results of this test were intended to give insight into the dependence of the truncation error on the accuracy of the approximation to the first derivative.

The test results for both truncation error tests is given below.

X	Test 1	Test 2	Exact
0.0	0.00000	0.00000	0.00000
2.0	84.296	84.375	84.717
4.0	470.82	470.81	471.30
6.0	1749.5	1749.5	1750.0
8.0	4873.2	4873.2	4873.7
10.0	11118.	11118.	11118.
12.0	21705.	21705.	21706.
14.0	35013.	35013.	35013.
16.0	25932.	25932.	25932.

The table shows that the results agreed to was very good for values of x that were removed from the origin. The answers agree to 5 significant digits with both the exact answer and each other. On the other hand, the points near the origin show only 2 or 3 significant digits in agreement. Furthermore, the difference in the first derivatives between the first and second truncation test are very small.

To determine why the agreement was bad near the origin, the program was modified to include the 'calculated' first derivative of the function on the mesh. Except in the neighborhood of zero, the approximation to the first derivative was quite close to the actual first derivative. Even in the neighborhood of zero, the accuracy was quite good. This difference is due to the numerical instability in the recursion relation for the k's. The difference did not get very large at zero.

The difference in the agreement of the answers in x can be explained by noting that as x gets larger in value, the regions in the mesh removed from zero contribute more heavily to the final answer. Since the spline approximation is more accurate there, the agreement becomes significantly better.

This low amount of truncation error observed, even when the mesh size is moderately large is encouraging. Of course, approximation of polynominals on closed intervals by polynomials of lower order is a reasonable approximation process [10].

8.3. The Tail Test

However, the good uniform approximation observed above for polynomials need not be true in the case where the forcing function is not close to being a polynomial on the mesh. In particular, section 4 showed that a reasonable portion of the forcing function on the mesh will be a term that decays with the axial distance from the origin at some finite power (as opposed to exponentially). Therefore, the next set of tests were carried out to determine the accuracy of the approximation in just such a case.

The third test using known forcing functions was based on using a simplified form of the tail behavior as the forcing function both on and outside the mesh. The idea here was to gain a better idea of how to judge the truncation error importance for the more reasonable class of forcing functions with polynomial decay. Note that the functions with compact support yielded an exponential rate of decay which is far too fast for the rate of decay that was estimated earlier.

The logical candidate for the forcing function was a member of the class

$$\phi(\xi) = c_1 |\xi|^{-\alpha_1} \qquad \xi < 0 \qquad (8.3.1a)$$

$$= c_2 \xi^{-\alpha_2} \qquad c < \xi \qquad (8.3.1b)$$

Since the singularities at the origin are too strong, then the integral with the Greens function does not exist in the classical sense. To avoid this problem, it was decided to alter the forcing function to be nonsingular in a neighborhood of the origin. One such class of modified forcing functions have the particular form,

$$\phi(\xi) = (-\xi)^{-\alpha_1} \qquad \xi < -\xi^* \qquad (8.3.2a)$$

$$= (-\xi^*)^{-\alpha_1}\left[1 - \alpha_1 c_1 \eta + \tfrac{1}{2}\alpha_1(\alpha_1+1)c_2\eta^2 - \tfrac{1}{6}\alpha_1(\alpha_1+1)(\alpha_1+2)c_3\eta^3\right] \qquad -\xi^* < \xi < 0 \qquad (8.3.2b)$$

$$= (\xi^*)^{-\alpha_2}\left[1 - \alpha_2 c_1 \eta + \tfrac{1}{2}\alpha_2(\alpha_2+1)c_2\eta^2 - \tfrac{1}{6}\alpha_2(\alpha_2+1)(\alpha_2+2)c_3\eta^3\right] \qquad 0 < \xi < \xi^* \qquad (8.3.2c)$$

$$= (\xi)^{-\alpha_2} \qquad \xi^* < \xi \qquad (8.3.2d)$$

where, $\eta = (\xi/\xi^*) - 1$ (8.3.2e)

where the Boolean constants c1, c2, and c3 are used to fix the degree of the osculating approximation. If a constant is taken as one, all proceeding constants must also be one.

By construction, the modified forcing function must be at least globally continuous and can be made smoother by setting the higher Boolean constants in the set to one (along with all lower ones). There will be a discontinuity in one of the derivatives at the point where the two curves are matched, but the algorithm should be able to correctly identify the rate of decay since there are no other terms involved. The approximation at the origin is likely to suffer due to the discontinuity in the derivative, but a refined mesh should control the inevitable error. Thus, it is expected that a significant error will come from the intervals where the forcing functions are combined.

Furthermore, outside the neighborhood of zero, a truncation error will arise since no finite order polynomial could give an exact representation of the decay terms, (2a) and (2d) on the mesh; therefore, this test case will provide a more realistic estimate of the expected truncation error in this method.

Another parameter to be considered here is how wide the region about zero the neighborhood is taken to be. In the general case, recall the assumed conventions on the mesh

$$\xi_k = 0^- \quad (8.3.3a)$$

$$\xi_{k+1} = 0^+, \quad k = N/2 \quad (8.3.3b)$$

and by assumption of case,

$$\xi^* = \xi_{k+j} \quad 1 \le j \le \ell, \quad x = \xi_{k+\ell} \quad (8.3.3c)$$

there are only two general cases in integrating the forcing function (2c)-(2d) against the Green's function. It follows then that

$$\int_0^\infty G_\ell(\xi,x) q(\xi)\, d\xi = (\xi^*)^{-\alpha_2}\left[S_0(x,f) - c_1\alpha_2 S_1(x,f) + \tfrac{1}{2} c_2\alpha_2(\alpha_2+1) S_2(x,f) - \tfrac{1}{6} c_3\alpha_2(\alpha_2+1)(\alpha_2+2) S_3(x,f)\right] e^{-fx} + (\xi^*)^{-\alpha_2}\left[E_0(x,f) - c_1\alpha_2 E_1(x,f) + \tfrac{1}{2} c_2\alpha_2(\alpha_2+1) E_2(x,f) - \tfrac{1}{6} c_3\alpha_2(\alpha_2+1)(\alpha_2+2) E_3(x,f)\right]\sinh(fx) + FT1(\xi^*, \alpha_2, f)\sinh(fx) \quad (8.3.4a)$$

$$= (\xi^*)^{-\alpha_2}\left[S_0(x;f) - c_1\alpha_2 S_1(x;f) + \tfrac{1}{2} c_2\alpha_2(\alpha_2+1) S_2(x;f) - \tfrac{1}{6} c_3\alpha_2(\alpha_2+1)(\alpha_2+2) S_3(x;f)\right] e^{-fx} + FT2(x;\alpha_2,f,\xi^*) + FT1(x;\alpha_2,f)\sinh(fx) \quad (8.3.4b)$$

where the special functions FT1(x;α, f) and FT2 (x;α,f,X) are the same special functions introduced section 7.1 for the far field contribution, and the other special functions are defined by

$$S_i(x,f) \equiv \int_0^x \eta^i \sinh(f\xi)\, d\xi \quad (8.3.4c)$$

$$E_i(x,f) \equiv \int_x^{\xi^*} \eta^i e^{f\xi}\, d\xi \quad (8.3.4d)$$

and are defined only for this calculation.

To check the results of the tail test, a second exact calculation program was written to evaluate the integrals above. Since this test program, TEST2.FOR, was only to be run a few times to verify the accuracy of the main program, it was not necessary to hold the amount of computation in evaluating (7b) in section 7.1 to a minimum. Therefore, the two point Simpson's rule for the integral was replaced with a Romberg extrapolation to insure the necessary accuracy in the approximation of the integral.

The revised approximation is necessary because the reason for using the two point Simpson's rule is gone. The upper endpoint of the integrand instead of being on the order of a few per cent larger than the lower is now about 10 times the lower. That means that the integrand will vary significantly on the interval in question and a better approximation is necessary to get the same accuracy.

In the first attempt at the tail test, the results of the main code and TEST2 were in rough agreement, but not really close. Since the Green's function is the same in both cases, the real difference in the results had to come from either the integration procedures or the approximation of a function. It would be expected that both Gibbs' phenomena and the numerical instability in the k's would lead to a bad approximation of the chosen forcing function in the neighborhood of zero.

To test this hypothesis, a subroutine was added to the tail test procedure that would compare the spline approximation against the true forcing function on a chosen submesh. Statistics were recorded and presented on the maximum deviation, the maximum relative deviation, and the RMS of the fit.

The error behavior in the approximation is clearly bad. Moving the matching point back and/or increasing the degree of fit should improve the approximation; instead it got worst.

Unsmoothed K's
($\xi^* = 4$)

Constant

Mesh	Max Error	Max Rel Error	RMS
4.0	0.51281E-01	0.82050E+00	0.16197E-01
2.0	0.92229E-01	0.14757E+01	0.22426E-01
1.0	0.12391E+01	0.19826E+02	0.21771E+00

Linear Osculation

Mesh	Max Error	Max Rel Error	RMS
4.0	0.41092E-02	0.11352E+01	0.15009E-02
2.0	0.57498E+00	0.17408E+03	0.12728E+00
1.0	0.17536E+01	0.10021E+03	0.30813E+00

($\xi^* = 12$)

Constant Osculation

Mesh	Max Error	Max Rel Error	RMS
4.0	0.25781E-01	0.37125E+01	0.84176E-02
2.0	0.82304E+00	0.24918E+03	0.23288E+00
1.0	---	---	---

Linear Osculation

Mesh	Max Error	Max Rel Error	RMS
4.0	0.98271E+00	0.34733E+03	0.29172E+0
2.0	0.35569E+03	---	---
1.0	---	---	---

The error behavior in the approximation is clearly bad. Moving the matching point back and/or increasing the degree of fit should improve the approximation; instead it got worst.

A closer examination of the testing data showed that the relative error on the submesh demonstrated a pattern of oscillating growth, like the instability demonstrated in the calculations of the k's. Examination of the k's in these cases also showed the numerical instability encountered earlier.

The numerical evidence strongly suggested that modification of the forcing function excited the k-instability before it would have normally been noticeable. To reduce the error in the approximation, it was necessary to 'smooth' the k's to suppress a large portion of the instability induced error.

8.4. Smoothing the k's

Fortunately, the root of the recursion relation associated with this instability is negative. That means that the k's are continually over and under shooting the correct value of the first derivative. Following the example of the leapfrog scheme, if successive values could be combined, there would be subtractive cancellation of the error. The new point should be much closer to the desired curve than either of the points used to create it.

Returning to the analysis of the error propagation in the k's, it follows that expected rate of error growth is given by

$$\lambda = \sqrt{(1+\alpha_i)^2 - 1} + 1 + \alpha_i \qquad (8.4.2)$$
$$\alpha_i = h_{i+1}/h_i$$

and this gives a weighting factor for the averaging that should eliminate most of the instability error.

The averaged values can be used to refine the estimates of the k's by using a linear interpolation and calculating the value at the mesh point. To deal with the problem of the endpoint conditions, it was decided to use the first point (the endpoint of the mesh) as having an exact value, and

then estimate the value at the origin by using the spline approximation formula there. This procedure would then 'smooth' the estimates of the k's to something that would be closer to the true value of the first derivative on the mesh points. Otherwise, it would be necessary to return to a tridiagonal solution scheme for the k's and find a procedure to calculate the slope of the solution at the origin directly rather than as part of the solution.

This smoothing procedure was implemented in the code as a subroutine called by the subroutine that found the k's in the first place. The approximation test phase of the tail test was repeated again using the new procedure. The dramatic improvements in accuracy can be noted from the tables below.

Smoothed K's

Constant Osculation

Mesh	Max Error	Max Rel Error	RMS
4.0	0.13697E-01	0.42578E+00	0.49705E-02
2.0	0.66655E-02	0.15264E+00	0.19802E-02
1.0	0.29967E-02	0.57572E-01	0.56038E-03

Linear Osculation

Mesh	Max Error	Max Rel Error	RMS
4.0	0.68815E-01	0.20883E+01	0.21136E-01
2.0	0.13271E-01	0.80000E-01	0.30173E-02
1.0	0.72377E-03	0.16692E-01	0.16391E-03

(ξ * = 12)

Constant Osculation

Mesh	Max Error	Max Rel Error	RMS
4.0	0.59086E-02	0.20883E+01	0.17603E-02
2.0	0.90865E-02	0.27510E+01	0.20112E-02
1.0	0.95259E-04	0.16692E-01	0.21871E-04

Linear Osculation

Mesh	Max Error	Max Rel Error	RMS
4.0	0.20417E-02	0.10537E+01	0.60949E-03
2.0	0.34028E-03	0.34028E-01	0.79455E-04
1.0	0.59852E-04	0.16692E-01	0.99133E-05

As the tables above show, there is a significant reduction in the approximation error as a result of using smoothing. However, the reduction is not without a penalty, as the agreement actually gets worst if the mesh points are too far apart. Therefore, the choice of the proper mesh has been made more delicate since a too coarse mesh will lead to errors in the smoothing process. On the other hand, without the smoothing, the error in the approximation becomes unacceptable.

Once the accuracy of the spline approximation had been increased by smoothing, the question arises as to how accurate the computational scheme in the code actually happens to be. To explore this question, various mesh widths were used in the tail test and the test results tabulated against the answers provided by TEST2.FOR. The data is reproduced below.

($\xi^* = 4.0$, Constant Osculation)

X	4.0	2.0	1.0	TEST2
0.0	0.00000	0.00000	0.00000	0.00000
1.0			0.39158E-01	0.39102E-01
2.0		0.54257E-01	0.52962E-01	0.52790E-01
3.0			0.56406e-01	0.55932E-01
4.0	0.55714E-01	0.54592E-01	0.52839E-01	0.51939E-01
5.0			0.41682E-01	0.41101E-01
6.0		0.31881E-01	0.30798E-01	0.30605E-01
7.0			0.22831E-01	0.22769E-01
8.0	0.18992E-01	0.17334E-01	0.17295E-01	0.17276E-01
9.0			0.13451E-01	0.134462E-01
10.0		0.10715E-01	0.10732E-01	0.10731E-01
11.0			0.87581E-02	0.87583E-02
12.0	0.70529E-02	0.72766E-02	0.72857E-02	0.72860E-02
13.0			0.61594E-02	0.61598E-02
14.0		0.52740E-02	0.52784E-02	0.527910E-02
15.0			0.45757E-02	0.457726E-02
16.0	0.39746E-02	0.40037E-02	0.40056E-02	0.40057E-02

($\xi^* = 4$, Linear Osculation)

X	4.0	2.0	1.0	TEST2
1.0	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
1.0			0.87534E-01	0.87540E-01
2.0		0.98189E-01	0.10043E+00	0.10045E+00
3.0			0.86640E-01	0.86692E-01
4.0	0.55871E-01	0.65031E-01	0.65279E-01	0.65269E-01
5.0			0.46057E-01	0.46005E-01
6.0		0.32613E-01	0.32408E-01	0.32409E-01
7.0			0.23424E-01	0.23433E-01
8.0	0.17730E-01	0.17433E-01	0.17513E-01	0.17520E-01
9.0			0.13531E-01	0.13536E-02
10.0		0.10729E-01	0.10762E-01	0.10764E-01
11.0			0.87690E-02	0.87704E-02
12.0	0.70298E-02	0.72784E-02	0.72897E-02	0.72904E-02
13.0			0.61609E-02	0.61614E-02
14.0		0.52742E-02	0.52789E-02	0.52797E-02
15.0			0.45759E-02	0.45775E-02
16.0	0.39741E-02	0.40037E-02	0.40056E-02	0.40058E-02

(ξ * = 12.0, Constant Osculation)

X	4.0	2.0	1.0	TEST2
0.0	0.00000e+00	0.00000E+00	0.00000E+00	0.00000E+00
1.0			0.43897E-02	0.43897E-01
2.0		0.60046E-02	0.60046E-02	0.60046E-02
3.0			0.65986E-02	0.65986E-02
4.0	0.68177E-02	0.68172E-02	0.68171E-02	0.68972E-02
5.0			0.68973E-02	0.68972E-02
6.0		0.69267E-02	0.69262E-02	0.69260E-02
7.0			0.69353E-02	0.69350E-02
8.0	0.69680E-02	0.69380E-02	0.69345E-02	0.69335E-02
9.0			0.69228E-02	0.69203E-02
10.0		0.69136E-02	0.68876E-02	0.68807E-02
11.0			0.67907E-02	0.67719E-02
12.0	0.66812E-02	0.65704E-02	0.65089E-02	0.64756E-02
13.0			0.58821E-02	0.58612E-02
14.0		0.52117E-02	0.51764E-02	0.51685E-02
15.0			0.45382E-02	0.43525E-02
16.0	0.40556E-02	0.39952E-02	0.39918E-02	0.39909E-02

(ξ * = 12.0, Linear Osculation)

X	4.0	2.0	1.0	TEST2
0.0	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
1.0			0.12012E-01	0.12012E-01
2.0		0.15649E-02	0.15699E-01	0.15699E-01
3.0			0.16324E-01	0.16324E-01
4.0	0.15541E-01	0.15815E-01	0.15822E-01	0.15822E-01
5.0			0.14906E-01	0.14906E-01
6.0		0.13837E-01	0.13838E-01	0.13838E-01
7.0			0.12713E-01	0.12713E-01
8.0	0.11548E-01	0.11568E-01	0.11564E-01	0.11569E-01
9.0			0.10419E-01	0.10419E-01
10.0		0.92682E-02	0.92729E-02	0.92732E-01
11.0			0.81412E-02	0.81419E-02
12.0	0.70520E-02	0.70555E-02	0.70545E-02	0.70542E-02
13.0			0.60754E-02	0.60741E-02
14.0		0.52521E-02	0.52475E-02	0.52468E-02
15.0			0.45643E-02	0.45641E-02
16.0	0.40155E-02	0.40007E-02	0.40014E-02	0.40015E-02

As can be seen from the tables above, as the mesh sizes decreases, the estimates of the integral generally get better. It also shows that 'smoothing' the fit by making the forcing function continuously differentiable and/or making the neighborhood about the origin larger both increase the degree of agreement.

8.5. Repeat of the Previous Tests

Since the original test results were based on the previous 'unsmoothed' algorithm for calculating the k's, it is natural to ask how the test results would vary using the 'smoothing' algorithm. So, the calculations of the previous section were repeated.

The results are summarized in the table below. The mesh size was uniform and set to h=2.0.

	Constant		Linear		Quadratic	
X	Test	Exact	Test	Exact	Test	Exact
0.0	0.00000	0.000000	0.00000	0.000000	0.00000	0.000000
2.0	0.86466	0.864664	2.0436	1.999999	5.7292	5.72921
4.0	0.98168	0.981681	4.0058	3.99995	17.962	17.9625
6.0	0.99750	0.997499	6.0004	5.99961	379.88	379.885
8.0	0.99950	0.99497	7.9973	7.99715	659.51	659.507
10.0	0.99872	0.998715	9.9789	9.97893	1016.4	1016.40
12.0	0.99084	0.990836	11.844	11.8443	1433.4	1433.44
14.0	0.93233	0.932332	12.85	12.8497	1783.8	1783.76
16.0	0.50000	0.500000	7.5000	7.50000	1130.0	1130.0

	Cubic		Quartic	
X	Test	Exact	Test	Exact
0.0	0.00000	0.000000	0.00000	0.000000
2.0	18.081	19.9998	64.342	84.7171
4.0	87.725	87.9847	464.90	471.298
6.0	251.85	251.887	1745.5	1750.00
8.0	559.16	559.167	4869.4	4873.67
10.0	1053.8	1053.85	11114.	11118.2
12.0	1754.5	1754.52	21703.	21705.9
14.0	2492.2	2491.96	35022.	35013.2
16.0	1709.2	1709.00	25943.	25932.0

Comparing this table with the previous one, it is clear that test results differ more from the exact values with the 'smoothed' k's, especially near the origin and for a cubic and quartic polynomial. This is to be expected since the approximation of the k's were very good for the previous algorithm. However, the error is not that bad and can be reduced by decreasing the mesh size, as shown below.

	Cubic		Quartic	
X	Test	Exact	Test	Exact
0.0	0.00000	0.000000	0.00000	0.000000
2.0	19.970	19.9998	84.353	84.7171
4.0	87.981	87.9847	471.00	471.298
6.0	251.89	251.887	1749.7	1750.00
8.0	559.17	559.167	4873.4	4873.67
10.0	1053.8	1053.85	11118.	11118.2
12.0	1754.5	1754.52	21706.	21705.9
14.0	2492.0	2491.96	35013.	35013.2
16.0	1709.0	1709.00	25933.	25932.0

Therefore, the 'smoothed' algorithm does almost as well when compared to the exact results, provided that the mesh size is suitably small. It would appear that smoothing the k's leads to little additional error in the main line of the calculation.

9. Determining the Constants in the Tail

9.1. The Asympototic Behavior of the Solution

As noted in section 4, the approximation used for the forcing function is cubic in the 'near' field near the propulsor, and then has a tail in the far field that decays like an inverse polynomial. In particular, the following estimates were derived there for the tail

$$q(\xi) \simeq C_1(-\xi)^{-\alpha_1} \qquad \xi < -\Xi \qquad (9.1.1a)$$

$$\simeq \tilde{q}(\xi) \qquad -\Xi < \xi < \Xi \qquad (9.1.1b)$$

$$\simeq C_2(\xi)^{-\alpha_2} \qquad \Xi < \xi \qquad (9.1.1c)$$

where C_1, C_2, α_1, and α_2 are assumed to be given constants, and $\tilde{q}$ is a function defined over the half meshes as cubic polynomials.

Using the results on symmetry again, it is enough to carry out the calculation for the case on the positive half of the real line. Then, from the results in section (6.1) and (6.3), it follows that

$$\int_0^\infty G_+(\xi,x)q(\xi)d\xi = \int_0^\Xi G_+(\xi,x)\tilde{q}(\xi)d\xi + C_2\int_\Xi^\infty G_+(\xi,x)\xi^{-\alpha_2}d\xi \qquad x > \Xi \qquad (9.1.2)$$

Since the forcing function is given by a cubic polynomial over each of the interior intervals in the mesh, it follows in the case where the point x lies outside the mesh that

$$\int_0^\infty G_+(\xi,x)q(\xi)d\xi = \sum_{i<k+1}^{N} h_i\Big\{ q_{i+1}S_{00}(f\xi_i,fh_i) + q_iS_{01}(f\xi_i,fh_i) + h_i(t_i-d_i)S_{12}(f\xi_i,fh_i) \qquad (9.1.3)$$
$$- h_i(t_{i+1}-d_i)S_{21}(f\xi_i,fh_i)\Big\}e^{-fx} + C_2FT1(x;\alpha_2,f)\sinh(fx) + C_2FT2(x;\alpha_2,f)e^{-fx}$$

where the special integrals defined in Section (6) have been used to simplify the expression.

The first term which comes from the near field contribution clearly has exponential decay with respect to x, since the functions in the sum do not depend explicitly on x. Therefore, the asymptotic behavior of the solution in the far field must be determined by the behavior of the far field contribution, FT1 and FT2.

To estimate the rate of decay in x for FT1, note that by definition

$$FT1(x;\alpha,f) = f^\alpha\int_{fx}^\infty \frac{e^{-t}}{t^\alpha}dt \qquad (9.1.4)$$

Using the fact that $(1/x^\alpha)$ is a monotonically decreasing family of functions and the monotonicity property of the integral, it follows that

$$FT1(x;\alpha,f) \le \frac{1}{f^\alpha}\frac{1}{x^\alpha}\int_{fx}^\infty e^{-t}dt \qquad (9.1.5)$$
$$\le \frac{1}{f}\frac{1}{x^\alpha}e^{-fx}$$

Therefore, for x positive, the product can be bounded above by

$$FTI(x;\alpha,f)\sinh(fx) \le \frac{1}{2f}\frac{1}{x^{\alpha}} \qquad (9.1.6)$$

and it follows that the second term in (9.1.3) decays with x no slower than the original forcing function.

To show that the same result holds for the second term, construct the function

$$L(x) \equiv \int_{X}^{x} \frac{\sinh(\beta t)}{t^{\alpha}}\,dt \Big/ \frac{1}{x^{\alpha}}e^{\beta x} \qquad x>X \qquad (9.1.7)$$

and consider its behavior as x goes to infinity. Clearly, the value is indeterminate and calls for an application of L'Hopital's rule. This yields

$$\lim_{x\to\infty} L(x) = \lim_{x\to\infty} \frac{\sinh(\beta x)}{x^{\alpha}} \Big/ \frac{(\beta x-\alpha)}{x^{\alpha+1}}e^{\beta x} \qquad (9.1.8a)$$

$$= \lim_{x\to\infty} \frac{e^{-\beta x}\sinh(\beta x)}{\beta-(\alpha/x)} \qquad (9.1.8b)$$

$$= \frac{1}{2\beta} \qquad (9.1.8c)$$

Therefore, asymptotically, the numerator and the denominator have equivalent orders. The numerator is just the definition of FT2 (divided by some constants) and the denominator is the required rate of decay.

It follows that for large x, that the integral above asymptotically has the same order of polynomial decay as the forcing function, plus some terms that have an exponential decay.

9.2. Determining the Constants

The two remaining free constants in the cubic spline approximation are to be fixed by evaluating the the first and second derivatives of the tail at the edge of the mesh, and then requiring an approximation with two continuous second derivatives on each of the half meshes. This leaves the question of how to estimate the two parameters for the tail of the approximation.

Assuming that the forcing function belongs to the class of functions that decay like $x^{-\alpha}$, it is possible to construct an estimate of the coefficients in the tail by using the value at three points that are "sufficiently" removed from the origin.

Assume that x, y, and z are three points near the edge of mesh, and

$$f_1 = q(x) \qquad (9.2.2a)$$

$$f_2 = q(y) \qquad (9.2.2b)$$

$$f_3 = q(z) \qquad (9.2.2c)$$

are three known values. Then, the approximation to the tail is given by the formulas

$$\alpha \simeq \frac{1}{2}\left[\frac{\ln(f_2/f_1)}{\ln(x_1/x_2)} + \frac{\ln(f_3/f_2)}{\ln(x_2/x_3)}\right] \qquad (9.2.3a)$$

$$C \simeq \frac{1}{3}\left[f_1|x|^{\alpha} + f_2|y|^{\alpha} + f_3|z|^{\alpha}\right] \qquad (9.2.3b)$$

This estimate was implemented in the subroutine that determined the interpolation parameters. This was done by modifying the program to allow calculation of the value of the function to be known at several points beyond the edge of the mesh as well as at the edge of the mesh itself.

Since inverse polynomial decay does not allow for a change of sign, the values were checked to be sure that they all had the same sign. If not, the program issues an error message that the mesh is probably too short, and stops.

At this point, the approximation to the first and second derivative of the tail at the edge of the mesh can be used to start the calculation of the k's by using the recursion relationship. Once the k's have been determined, then the integral of the Green's function with the forcing function can be determined for points belonging to the mesh, as well as points lying beyond the edge of the mesh by the formulas presented earlier.

9.3. Summary of One-Dimensional Problem

Therefore, at this point, an algorithm has been established and coded numerically to solve the one dimensional ordinary differential equation given by (2.7) provided that the forcing function $q(x)$ belongs to the class of functions that have multi-pole behavior sufficiently far away from the origin. The solution

1 has multi-pole behavior far from the origin, and the order of the pole will be no less than that of the forcing function,

2 has continuous second derivatives on the negative and positive half lines,

3 has a specified jump at the origin.

The original computer algorithm had shown in Section 8 that it could accurately reproduce results for the case of forcing functions that are cubic near the origin, and zero past some finite point that serves as the edge of the mesh. The truncation error is acceptable, and the accuracy of the results did not appear to be sensitive to small errors in the specification of the first derivative at the edge of the mesh.

However, when the tail test was done, using a modified function with multi-pole behavior far from the origin that was finite at the origin; the accuracy of the numerical solution vanished near the origin due to rapid error growth in the procedure to estimate the first derivative in the spline approximation.

Since the error growth was oscillatory, and the rate can be estimated locally from a second order finite difference equation, the solution procedure was modified to 'smooth' the estimates of the k's before using them to calculate the integrals. This smoothing procedure reduced the accuracy somewhat in the cases of "nice" test functions, but reduced the numerical instability in the calculations of the k's to the point where the estimates of the solution near the origin were reasonably accurate again.

10. Closing the Loop

In Section 1, Goodman's linear partial differential equation for the transformed pressure and velocity, was converted into a "quasi" ordinary differential equation that was linear. The Green's function for this system was derived, and the algorithm for solving the one dimensional problem was coded and tested in the previous sections.

The next step in the problem arises from the fact that the forcing function in this linear "ODE" is actually a functional acting on the solution itself, turning the expression for the solution of the "ODE" into an integral equation.

Therefore, the method of successive approximations is to be made to solve the resulting linear equation. An initial approximation to the forcing function is used in the integral form of the solution of the "ODE" to yield a new estimate of the solution that satisfies both the far field conditions as well as the jump condition at the origin. This new approximation of the solution can be used to form an improved estimate of the forcing function.

At this point, the loop is closed since the new estimate of the forcing will yield another estimate of the forcing function, and so forth. The implicit assumption to this procedure is that the successive iterations of the formula

$$\rho_m^{(l)}(x;r) = \tfrac{1}{2}\Delta\rho_m(r)e^{-x f_m(r)} - \frac{1}{f_m(r)}\int_c^\infty G_+(\xi,x)\{\Theta(\rho_m^{(l-1)}) + \Re(\rho_m^{(l-1)})\}(\xi,r)\,d\xi \qquad (10.1)$$

$$x>0,\quad l=1,2,\cdots$$

$$\Re(\rho_m)(\xi;r) = -\frac{1}{r}\frac{\partial}{\partial r}r\frac{\partial}{\partial r}\rho_m(\xi,r) \qquad (10.2)$$

$$\Theta(\rho_m)(\xi,r) = \sum_{j=-\infty}^{\infty} v_{m-j}(r)\rho_j(\xi;r) - v_0(r)\rho_m(\xi;r) \qquad (10.3)$$

will converge in some suitable norm to a suitable function that will be the solution of the integral equation and the equivalent differential equations.

Some of the issues involved in constructing a convergent iteration procedure will be discussed in a later section. In the next section, the extension of the algorithm to include the radial dependence will be discussed. It is possible to separately include the radial and harmonic effects in this fashion because the original system is linear.

10.1. Extension to Include Radial Effects

As noted above, the operator that changes the result of the integration into the forcing function for the next step is an operator that

can be decomposed into a radial operator and a harmonic interaction operator. Consider the radial part of the operator given by

$$\mathcal{R}(\rho_m) \equiv \frac{1}{r}\left(\frac{\partial}{\partial r} r \frac{\partial}{\partial r}\right)\rho_m \qquad (10.1.1)$$

The information about the function in question consists of a discrete grid of values giving a dependence in the axial and radial directions. In order to calculate the derivative, it is necessary to generate an approximate curve in the radial direction just as was done in the axial direction. Since the axial representation in the mesh is done using splines, it seems reasonable to use the same representation in the radial one as well.

Assuming a spline representation in the form

$$\rho_m(r) = s\rho_m^{i+1} + (1-s)\rho_m^{i} + \varrho_i s(1-s)\left[(\tilde{k}_i - \tilde{d}_i)(1-s) - (\tilde{k}_{i+1} - \tilde{d}_i)s\right] \qquad r_i \le r \le r_{i+1} \qquad (10.1.2a)$$

where,

$$s = (r - r_i)/\varrho_i \qquad r_i \le r \le r_{i+1} \qquad (10.1.2b)$$

$$\tilde{d}_i = (\rho_m^{i+1} - \rho_m^{i})/\varrho_i \qquad (10.1.2c)$$

$$\varrho_i = r_{i+1} - r_i \qquad (10.1.2d)$$

$$\rho_m^{i} = \rho_m(r_i) \qquad (10.1.2e)$$

then, using the chain rule, the first derivative with respect to r is given by

$$\frac{\partial}{\partial r}\rho_m(r) = \tilde{d}_i + (1-s)^2(\tilde{k}_i - \tilde{d}_i) + 2s(1-s)(\tilde{d}_i - \tilde{k}_{i+1}) + s^2(\tilde{k}_{i+1} - d_i) \qquad (10.1.3)$$

Multiplying by r, taking another derivative, and then dividing by r gives the expression

$$\mathcal{R}(\rho_m)(r) = \frac{\tilde{d}_i}{r} + \frac{1}{r}\left\{(1-s)^2(\tilde{k}_i - \tilde{d}_i) + 2s(1-s)\left[2\tilde{d}_i - \tilde{k}_i - \tilde{k}_{i+1}\right] + s^2(\tilde{k}_{i+1} - \tilde{d}_i)\right\}$$
$$+ 2\varrho_i\left\{(1-2s)\left[2\tilde{d}_i - \tilde{k}_i - \tilde{k}_{i+1}\right] + s(\tilde{k}_{i+1} - \tilde{d}_i) - (1-s)(\tilde{k}_i - \tilde{d}_i)\right\} \qquad (10.1.4)$$

Specializing the result for the two endpoints of the interval, then

$$\mathcal{R}(\rho_m)(r_i) = \frac{\kappa_i}{r_i} + 2\varrho_i\left\{3\tilde{d}_i - 2\tilde{k}_i - \tilde{k}_{i+1}\right\} \qquad (10.5)$$

$$= \frac{\kappa_{i+1}}{r_{i+1}} - 2\varrho_i\left\{3\tilde{d}_i - \tilde{k}_i - 2\tilde{k}_{i+1}\right\} \qquad (10.6)$$

Therefore, given the value of the function on the cross-mesh and the spacing of the node points, calculation of the spline coefficients is enough to allow estimation of the operator that turns the radial component into a forcing function.

The remaining point that needs to be resolved at this point in the radial spline approximation algorithm is how to determine the two remaining free parameters to get closure of the linear system for the values of the first derivative. The first derivative is not known at either the hub, or infinity, so supplying the values for the first derivatives at the end of

the mesh is not practical. On the other hand, the behavior in the far field is also not known, so the previous scheme of matching to the far field tail has its problems as well.

Therefore, the condition that is imposed at the far boundary is that the second derivative at the end points should vanish. This gives a tridiagonal system to be solved for the k's which can be done in an efficient fashion.

To test the accuracy of this algorithm, a small test program was constructed that used the proposed spline approximation and the formula for calculating the radial forcing contribution. The program used as its test values polynominals of order three or less which have exact representations, and yield a known forcing response. As expected, the results were very good at the interior of the mesh.

However, on the endpoints of the mesh, the approximation was very bad, as reflected in the table below. The mesh patterns (1, 2, 3, and 4) refer to how the mesh points were distributed over the interval.

Quadratic Forcing

Mesh Pattern	Mean Square Forcing (Old)	Mean Square Forcing (New)
1	5724.6	17.65
2	3118.1	19.16
3	1240.9	17.68
4	592.0	16.87

The mean square error in the calculation of the derivatives is quite acceptable, but the mean square error in the forcing term is overly large, about the magnitude of the terms themselves. This is almost solely due to the bad approximation of the true forcing function at the end point of the intervals.

Examining the expression for the forcing function more closely, the term for approximating the second derivative has the same linear combination that is used at the mesh points to make the second derivative vanish. As a result, the approximation of the forcing function at the mesh points consists solely of the first derivative divided by r, an inadequate approximation as can be seen from the table above.

Therefore, it was necessary to find an alternative set of end conditions that could be applied and would yield a more accurate approximation of the true second derivative at the ends of the interval. As an alternative set of conditions, the end conditions from problem 4.6.4 of Dahlquist

$$\tilde{K}_2 - \tilde{K}_1 = \frac{1}{4\rho_1}(\rho_m^1 + \rho_m^2 - 2\rho_m^{1\frac{1}{2}}) \qquad (10.1.7)$$

$$\tilde{K}_M - \tilde{K}_{M-1} = \frac{1}{4\rho_{M-1}}(\rho_m^M + \rho_m^{M-1} - 2\rho_m^{M-\frac{1}{2}}) \qquad (10.1.8)$$

where the extra values are defined by

$$\rho_m^{1\frac{1}{2}} = \rho_m\left(\tfrac{1}{2}(r_1 + r_2)\right) \qquad (10.1.9a)$$

$$\rho_m^{M-\frac{1}{2}} = \rho_m\left(\tfrac{1}{2}(r_M + r_{M-1})\right)$$

give a fourth-order accurate approximation to the function in question, provided that (10.1.9a-b) are given and the third derivative is continuous at these points.

To approximate the value of the function at these points, use a three point Lagrange interpolation formula at the end point of the mesh. (Clearly a linear interpolation scheme would not provide any information about the behavior of the second derivative near the endpoint.) In the notation used so far, it follows that

$$\rho_m^{1\frac{1}{2}} = \tfrac{1}{2}\left[\rho_m^{1}\frac{(r_3-r_{1\frac{1}{2}})}{\ell_1+\ell_2} + \rho_m^{2}\frac{(r_3-r_{1\frac{1}{2}})}{\ell_2} - \tfrac{1}{2}\rho_m^{3}\frac{\ell_1^2}{(\ell_1+\ell_2)\ell_2}\right], \qquad r_{1\frac{1}{2}} = \tfrac{1}{2}(r_1+r_2) \qquad (10.1.10a)$$

$$\rho_m^{M-\frac{1}{2}} = \tfrac{1}{2}\left[-\tfrac{1}{2}\rho_m^{M-2}\frac{\ell_{M-1}^2}{\ell_{M-2}(\ell_{M-2}+\ell_{M-1})} + \rho_m^{M-1}\frac{(r_M-r_{M-\frac{1}{2}})}{\ell_{M-2}} + \rho_m^{M}\frac{(r_{M-\frac{1}{2}}-r_{M-2})}{\ell_{M-2}+\ell_{M-1}}\right], \quad r_{M-\frac{1}{2}} = \tfrac{1}{2}(r_M+r_{M-1}) \qquad (10.1.10b)$$

which are easy enough to calculate.

The test case was run again with the new end conditions applied (note that a tri-diagonal system is still the result), and the mean square errors calculated again. The results for the forcing function are summarized in the table above under the new column.

10.2. Extension to Include Angular Effects

In the previous section, the extension of the code to handle the inclusion of the radial dependence of the solution in the iteration procedure was detailed. To complete the forcing term, it remains to add the terms that give the angular dependence to the the calculation of the forcing function.

The angular part of the forcing function is given by the functional

$$\Theta(\rho_m)(r) = \sum_{j=-\infty}^{\infty} v_{m-j}(r)\rho_j(r) - v_c(r)\rho_m(r) \qquad (10.2.1)$$

This form is particularly simple since the Fourier analysis has already been done explicitly in the early part of the derivation.

This term represents the transfer of energy from one mode of the velocity to another mode in the pressure. In the program, it is simply modeled by calculating the value of (10.2.1) based on the value at the previous iteration. The initial values, of course, come from the initial estimate of the solution.

11. Thoughts on Convergence

In order for the iterative equation to be a convergent iterative scheme under the method of successive approximation, it is necessary that [12].

(1) The mapping be from a subspace into the same subspace

(2) In a reasonable norm for that subspace, the mapping should be coercive or compact.

should be coercive or compact.

(3) The limit of the sequence of successive approximations exist in the subspace.

The first point means that the iteration can be continued at each step in the calculation. The second point is necessary if the iteration is to converge. The third point means that if the iteration does converge, then the limit of the sequence exists in the same subspace, ie. the subspace is closed under the appropriate norm.

These three questions could be answered by a detailed study of the kernel for the integral equation. However, the kernel for this equation is very complicated since it is the Green's function found earlier (2.6) multiplying the radial and harmonic operators acting on the pressure. Establishing the properties of the Green's function is rather simple, but when the radial and harmonic operators are considered, analytic study becomes very difficult. This is due to the fact that these operators map functions of function spaces into other function spaces.

The first step, is to determine the domain of the combined operators. One of the obvious properties that can be required is that solution must go to zero sufficiently quickly with the axial distance that the integrals with the Green's function will not be divergent. If the functions are roughly separable, then the behavior of the harmonic and radial operators will not change the behavior in the axial direction.

The first choice, the subspace of functions with compact support is clearly not appropriate because the image is no longer in that space. In the process of carrying out the calculations in section (9.1), it was demonstrated that functions that were defined on the mesh, and then were zero outside, decayed exponentially with axial distance from the origin. Since the choice of the mesh is reasonably arbitrary and the exact nature of the continuous function on the mesh was not used, it follows that the same types of estimates would be derived for functions with compact support.

So, it follows that the far field contribution will have exponential decay for functions with compact support, and clearly the image will not be a member of the class of functions of compact support.

The result that functions of compact support were mapped to functions that decayed exponentially, suggested that perhaps the proper subspace could be the functions with exponential decay. However, a quick calculation of the far field contribution similar to the one done in section (9.1) demonstrated that rate of decay of the image function could be smaller than the rate of decay for the argument function.

Although we now have a function space that has the desirable property that that under the functional operator associated with the integeral equation for this problem, the space maps into itself; the fact that the mapping does not appear to lead to at least the same rate of exponential decay suggests that the operator as a mapping from the space of exponentially decaying functions into itself is neither coercive nor compact, which is the second required condition for the iterative procedure to converge. In fact, it would appear possible that a function could be

found that under successive mappings would actually leave the space of exponentially decaying functions in the limit as the number of iterations approaches infinity, suggesting that the operator is not closed on this space.

Finally, the initial decay rate estimates in section (4) and (9.1) suggest that functions with inverse power decay are more appropriate for the problem at hand, since our initial approximation will come from that class.

To investigate this possibility further by extending our estimates of the operator, reconsider the calculations done in the analysis of the generalized tail test in section (9.1)

$$FT1(x;\alpha,f)\sinh(fx) \le \frac{1}{2f}\frac{1}{x^\alpha} \qquad (11.1)$$

$$FT2(x;\alpha,f)e^{-fx} \simeq \frac{1}{2f}\frac{1}{x^\alpha} \quad x \gg \Sigma \qquad (11.2)$$

Sharping the estimates given there slightly, it quickly follows that

$$\int_0^\infty G_+(\xi,x)q(\xi)d\xi = \frac{1}{2}\frac{2f-\frac{\alpha}{x}}{f(f-\frac{\alpha}{x})}\frac{1}{x^\alpha} + h.o.t. \quad x > \Sigma \qquad (11.3)$$

and this term clearly remains in the subspace with the same rate of multi-pole decay as the original forcing function.

It follows then from the above results that the integral will map the subspace of functions that decay like x to the α power into itself. This provides the first necessary condition for the method to converge.

On the second necessary condition for the method to converge, it should be noted that for the field point sufficiently large, then

$$\int_0^\infty G_+(\xi,x)q(\xi)d\xi \simeq \frac{1}{f}\frac{1}{x^\alpha} + O\left(\frac{1}{x^{\alpha+1}}\right) \qquad (11.4)$$

and the image function has the same multi-pole dominant behavior as the argument function. In addition, if the parameter f is greater than one, then it becomes possible that image function will have a smaller norm than the argument, making the integral operator a contraction mapping in this space.

Showing that the operator is a contraction mapping in the space in question would be a definite step forward, but would not be enough to conclude that it was coercive or compact. The space is question is a function space, and it is possible to construct contraction mapping in a function space that do not converge (this does not happen in a finite dimensional space). The only point remaining would be to establish that the operator was sufficiently restricted dimensionally and closed so that sequence formed by successive iterations converged to a function in the space.

To proceed further analytically in this matter requires specification of a proper norm for this function space. Once the norm has been specified,

then the requirements for this subspace to be closed and the mapping to be coercive or compact can be explored by rather technical analytical methods. With these results in hand, then the more practical questions about the domain of attraction, rate of convergence, and stability of the iteration procedure can be considered in a rational matter.

Our conditions on the function at this point include that the space in question consist of functions which will have continuous second derivatives in the independent variables in the region excluding the disk representing the propeller. At the propeller, it is necessary to allow the function to jump in order to satisfy the boundary conditions imposed by the pressure jump condition. In addition, the function must decay at least as fast as some multi-pole in the axial distance.

Putting all these requirements together in a complete description of of a function space with the appropriate norm to establish the remaining conditions for successive iteration to succeed in this case requires a great deal of ingenuity and insight. Unfortunately, the insight necessary on this point did not come during this phase of work.

Therefore, these issues in the analytical analysis of the problem will bypassed in order to study how the iteration procedure performs. That behavior will be taken as indicative of the results of doing the analytical analysis of convergence for the iteration scheme.

12. Convergence Tests

Once the one dimensional code was up and running, it was time to explore what the integral operator would do upon iteration. As noted in the previous section, the integral operator has the property that it maps the subspace of functions with multi-pole decay rates into itself. The two questions that are to be explored by numerical experimentation are

(1) Is the iteration stable?

(2) Is the result unique?

To explore these issues numerically, the tail test was run inside a larger loop so that the result at the ninth stage would be used as the approximation to the forcing function at the (n+1)st stage in the calculation. This calculation corresponds to the attempted solution of the homogenuous integral equation.

It has already been suggested in the previous section that the operator may be a contraction operator in a suitable normed function space. The more desirable property would be that the operator is a strict contraction operator, ie. that the operator norm is less than one, because this would insure that the region in the domain that could contain a fixed point will shrink in volume at each and every iteration.

Should the numerical iterations begin to drift to zero, then there is a region of the subspace about the origin in which the integral operator is a strict contraction operator. Since the original equation is linear, this implies that there will be a region about the solution in which the iteration sequence should converge.

There will be no guarantee that the sequence will converge to a function in the subspace (that problem can be resolved by looking at the associated weak problem and its closure of the function space, given a natural normed linear function space for this problem). Establishing a candidate for the solution numerically will be the chief goal of the remaining work.

A second possibility that must be considered is that iteration sequence will not converge to 0, but appears to be approaching a non-zero function. The difficulty here is that any such function is likely to be an eigenvalue of the integral operator. Once the integral operator is known to have one or more eigenvalues, then uniqueness of the solution becomes a real problem and solving the system by successive iteration becomes extremely difficult.

12.1. Results of the Convergence Test

The one dimensional code was modified by adding an iteration loop where the output from the integration was used as the input data for the next iteration. The initial forcing function used was the generalized tail test function mentioned earlier, in part because the code already existed for using it, and for checking the current algorithm by using the fact established above that the 'polynomial' order of the forcing function would be preserved by the operator associated with the integral.

The first test was very disappointing as an instability arose in the determination of the rate of decay. Both the coefficient multiplying the function and the estimate of the polynomial order began to show an oscillating instability.

Since it was already established that the polynomial order of the forcing function would be preserved under the operator (11.4), the subroutine that finds the k's was modified to find the rate of decay only on the initial input data. On subsequent iterations, the rate of decay was specified by as the initial rate, and the best least squares estimate of the coefficient was determined.

The results of the iteration calculation are summarized in the table below. The parameters that were varied were the step size and the the extent of the mesh.

H = 2.0, X = 16.0

X	Original	1	2	3	4
0.0	0.18750E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
2.0	0.12500E+00	-0.98189E-01	0.73150E-01	-0.57705E-01	0.47360E-01
4.0	0.62500E-01	-0.65031E-01	0.63543E-01	-0.58871E-01	0.53631E-10
6.0	0.27778E-01	-0.32613E-01	0.36129E-01	-0.38092E-01	0.38643E-01
8.0	0.15625E-01	-0.17433E-01	0.19537E-01	-0.21555E-01	0.23247E-01
10.0	0.10000E-01	-0.10729E-01	0.11622E-01	-0.12655E-01	0.13749E-01
12.0	0.69444E-02	-0.72784E-02	0.76770E-02	-0.81497E-02	0.86945E-01
14.0	0.51020E-02	-0.52742E-02	0.54714E-02	-0.57009E-02	0.59685E-02
16.0	0.39063E-02	-0.40037E-02	0.41169E-02	-0.42503E-02	0.44069E-02

H = 1.0, X = 16.0

X	Original	1	2	3	4
0.0	0.18750E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
2.0	0.12500E+00	-0.10043E+00	0.76414E-01	-0.59012E-01	0.47100E-01
4.0	0.62500E-01	-0.65279E-01	0.63273E-01	-0.58787E-01	0.53518E-01
6.0	0.27778E-01	-0.32408E-01	0.35753E-01	-0.37568E-01	0.38093E-01
8.0	0.15625E-01	-0.17513E-01	0.19553E-01	-0.21454E-01	0.23027E-01
10.0	0.10000E-01	-0.10762E-01	0.11681E-01	-0.12705E-01	0.13760E-01
12.0	0.69444E-02	-0.72897E-02	0.77042E-02	-0.81906E-02	0.87392E-02
14.0	0.51020E-02	-0.52789E-02	0.54835E-02	-0.57219E-02	0.59976E-02
16.0	0.39063E-02	-0.40056E-02	0.41212E-02	-0.42558E-02	0.44122E-02

H = 2.0, X = 24.

X	Original	1	2	3	4
0.0	0.18750E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
2.0	0.12500E+00	-0.98189E-01	0.73150E-01	-0.57705E-01	0.47360E-01
4.0	0.62500E-01	-0.65031E-01	0.63543E-01	-0.58871E-01	0.53631E-01
6.0	0.27778E-01	-0.32613E-01	0.36129E-01	-0.38092E-01	0.38643E-01
8.0	0.15625E-01	-0.17433E-01	0.19537E-01	-0.21555E-01	0.23247E-01
10.0	0.10000E-01	-0.10729E-01	0.11622E-01	-0.12655E-01	0.13749E-01
12.0	0.69444E-02	-0.72785E-02	0.76774E-01	-0.81502E-02	0.86947E-02
14.0	0.51020E-02	-0.52751E-02	0.54729E-02	-0.57017E-02	0.59661E-02
16.0	0.39063E-02	-0.40049E-02	0.41138E-02	-0.42354E-02	0.43725E-02
18.0	0.30864E-02	-0.31469E-02	0.32121E-02	-0.32828E-02	0.33602E-02

If the previous estimates of the 'polynomial order' are examined again, it suggests that in magnitude, the image of a generalized tail test function asymptotically, is the same generalized tail test function with a different coefficient. Closer examination of the iteration results in the table above shows that there is not much variation from the value of the forcing function and the result other than the sign. There is a tendency shown for the image to be less than the forcing function in a neighborhood of the origin, and somewhat greater in the far field. Still, the values do not appear to be that different.

This is expected from the previous results, and there is even a prediction that the coefficient should be approximately 1/f in magnitude. (There are, after all, functions of higher polynomial order that result from the integration.) A comparison was made of the change in the coefficient as both the iteration number and the mesh parameters were changed.

H	X	Original	1	2	3	4
2	16	1.00000	-1.03692	1.07403	-1.11523	1.16219
1	16	1.00000	-1.02624	1.05681	-1.09234	1.13354
2	24	1.00000	-1.02699	1.04709	-1.06564	1.08399
1	24	1.00000	-1.01117	---	---	---

The variation in the constants is nearly linear with iteration number, and is reduced by decreasing the mesh size and/or increasing the extent of the mesh. There are practical limits in the improvement process, as reflected in the missing entries in the table. When the extended mesh was

used with the fine mesh size, the numerical instability magnified the round-off and truncation errors to the point that the results were no longer symmetric about the origin. Therefore, there was little use in continuing the test in a regime where numerical errors were dominant.

The results are encouraging over those of the original procedure when the polynomial order was redetermined each and every time through the loop. However, a two per cent relative error at each stage of the calculation (or more) is clearly a problem. That places a definite limit on the number of iterations that can be made to improve the solution before error growth will become dominant. It would clearly be desirable to reduce the error induced at each stage of the iteration.

As a first step in this direction, the code was modified to allow the user to specify how many points would lie outside the mesh. There were several reasons why this modification was tried. First, there is a clear problem in reducing the mesh size to the point of having more than 50 points in the mesh. The numerical instability in the calculation of the k's includes a significant amount of round-off and truncation error. Smoothing will not control this, only extended precision in the calculation will help. Second, the generalized tail test calculation shows that there is generally a term of exponential order, and terms of higher polynomial order in the result of the integration. By taking more points outside the mesh, it is hoped that a better estimate of both the true polynomial order of the integral and the associated coefficient will be obtained. In this way, the error growth could be reduced to an acceptable level for the purposes of the computation at hand.

Repeating the calculations using the extended mesh with the additional points on the outside of the mesh did raise the accuracy of the points at the edge of the mesh, and reduced the initial growth rate in the estimate of the coefficient. However, the coefficient for the tail test function still grew. Three methods were used to control it, first to extend the mesh, second refine the mesh, and third, add exterior points to the mesh. The latter two modifications did tend to improve the estimate of the coefficient under iterations. However, the reductions were slight. A more useful reduction resulted by extending the mesh.

An attempt was made to both extend the mesh and refine it, but the round-off error in the calculations of the k's caused the symmetry to be broken and the calculation to break down.

13. The Iteration Breakdown

In the results of the previous section, there is no indication that the sequence of iterations is trying to converge to zero, in fact, it appears that the iteration procedure is attempting to converge to a function that is very close to the generalized tail test function, which itself was chosen because it has the correct asymptotic behavior at far downstream and upstream (Section 1.4).

The previous analysis with the generalized tail test functions also suggests that in the asymptotic limit, that functions with multi-pole behavior at infinity are the eigenvalues of the associated integral operator.

These functions (and there is an infinite number of them) have the potential to 'drown out' the result under an iterative method since repeated iteration with the integral operator will tend to introduce them into the calculation via the smallest amount of numerical noise. This makes the calculation rather dubious.

It is true that the iteration is not done in the simple fashion that was done here. There is an operator that takes the result and turns it into the forcing function for the next stage of the calculation. However, that operator does not depend directly on the axial coordinate (for that was how the splitting was arranged in the first place!), and would not suppress or eliminate these functions.

Therefore, it is necessary on the basis of numerical experiment and analytic results that this iteration procedure is not suitable for solving the problem at hand since it lacks stability and the associated homogenous integral equation has solutions that appear to be in the same class as the physical solution that we seek.

The traditional, analytic solution to this difficulty would be to split the proposed solution into two parts, a solution that is spanned by the basis of the eigenvalue subspace, and a solution that is in the perpendicular complement of the eigenvalue subspace. The operator would also be split into the sum of an operator acting on the eigenspace, and another operator which acts on the perpendicular complement of the eigenspace.

This problem could be solved in theory as long as the space of eigenvalues was finite dimensional, and the operator on the prependicular complement (which is infinite dimensional) has an inverse on that space.

This coefficients must be determined by adding auxiliary conditions that are associated with the 'physical' nature of the problem and not reflected in the operator in question. Furthermore, there must one condition for each and every eigenvalue of the problem [13].

This approach was clearly not applicable to the problem at hand. As noted above, one of the most stringent requirements is that the number of troublesome eigenvalues must be finite for this decomposition to work. In the problem at hand, they are not.

14. Summary

The method of splitting the partial differential equation (0.7) into two parts appears to have worked reasonably well. The associated Green's function that was quite nice and required only moderate effort to program. Since the Green's function was a combination of elementary functions, the use of splines to approximate the solution in the near field lead to well known integrals that could be evaluated at the very beginning of the program for the contribution of each section on the mesh, and then stored for future calculation of the solution on that mesh.

The lack of explicit boundary conditions at both upstream and downstream infinity gave rise to a weak oscillating instability in the calculation of the remaining constants (k's) in the spline approximation of the solution and the forcing function. This instability was ameliorated and numerical accuracy maintained by the methods of (1) adjusting the grid size to be smaller near the origin where rapid changes occur, (2) expanding the grid spacing away from the origin to hold down the number of points actually need in the grid to reduce round-off error from exciting the instability, and (3) extending the mesh outside the region near the actuator where a better estimate of the multi-pole behavior could be obtained.

The numerical solution of the one dimensional problem proved to be straight forward after the difficulty with the weak instability in the calculation of the remaining spline coefficients was ameliorated.

However, when the method was extended to solve the problem in three dimensions, then the iteration method became unusable due to the presence of presumed eigenvalues of the associated integral operator. At this time, there is no good method for suppressing or eliminating them, so numerically the three dimensional problem is unstable.

15. Conclusions

The objective of this research was the development of a rational method for determining the spatial distribution of inflow to a propeller when operating in a non-axially symmetric hull wake that did not involve the numerically difficult task of iterating a series of five-fold integrals over the entire space.

The method chosen was to assume that the physics of the problem would be dominanted by the flow in the longitudinal direction, and that the lateral flow would be a first order correction. This allowed the pressure equation to be rewritten as an integral equation with the radial and tangential effects included with an iteration procedure.

This method proved successful in testing on a collection of one-dimensional problems; but the iteration developed unexpected instabilities when tested on the three dimensional problem.

Further research into the literature has showed that this type of difficulty with handling inviscid shear flow by perturbation methods and/or the method of successive iteration was first noted by Lighthill [14]. He observed that the first approximation was of larger order at infinity than the base solution, and the attempt to use higher order corrections made the non-uniform convergence even worse.

It appears to be necessary to recast the problem in order to avoid these fundamental difficulties. One approach would be to include non-linear terms in the linearized steady Euler equations to reduce the interaction between the vorticity in the wake and the downwash. A second method would reduce the interaction by including the dissipation terms, which would lead to using Oseen's equations rather than the Euler equations.

Yet a third approach would be to recast the problem by decomposing it in the same fashion as done by Atassi [15] and Goldstein [16] in the case of a planar wing in a gust.

Further basic research is necessary to decide which of these three alternatives offer the best possibility of capturing the physics of the problem in a manner that can be used effectively.

REFERENCES

[1] Huang, T.T.,Wang,H.T.,Santelli,N., and Groves,N.C., "Propeller/Stern/Boundary Layer Interaction on Axisymmetric Bodies: Theory and Experiment," DTMB Report 76-0113, December 1976.

[2] Goodman, T.R., and Valentine, D.T. "Effective Inflow Velocities Into a Propeller Operating in an Axisymmetric Shear Flow", DL Report R-2129, November 1980.

[3] Goodman, T.R., "Momentum Theory of a Propeller in a Shear Flow", Journal of Ship Research, No. 4, December 1979, pp. 242-252.

[4] Belleni-Morante, A., "Applied Semigroups and Evolution Equations", Claredon Press, Oxford England, 1979.

[5] Tsakonas, S., Jacobs, W.R., and Ali, M.R., "Documentation of a Computer Program for the Pressure Distribution, Forces, and Moments on Ship Propellers in Hull Wakes", DL Report R-1863, January 1976.

[6] Tsakonas, S., Breslin, J.P., and Jacobs, W.R., "Blade Pressure Distribution for a Moderately Loaded Propeller", DL Report R-2063, September 1980.

[7] Abramowitz, M., and Stegun, I.A., "Handbook of Mathematical Functions", Dover Publications Inc., New York, NY., November 1970, formula 8.5.4.

[8] Abramowitz, M., and Stegun, I.A., formula 8.1.3.

[9] Abramowitz, M., and Stegun, I.A., formula 5.1.55

[10] Since this approximation is uniformly good over the interval, then the value of the integrals will not change by much either. The accuracy of the approximation is quite reasonable and not totally unexpected. More discussion on this point can be found in Section 4.2 of E. Issacson and H.B. Keller's "Analysis of Numerical Methods",John Wiley and Sons, Inc., New York, NY., 1966.

[11] Dahlquist, G., and Bjorck, A.,"Numerical Methods", Prentice-Hall, Inc., Englewood Cliffs, NJ., 1974.

[12] Larsen, R., "Functional Analysis", Marcel Dekker, Inc., New York, NY., 1973, Chapter 12.

[13] Pogorzelski, W., "Integral Equations and Their Applications", Pergamon Press, New York, NY., 1966.

[14] Lighthill, M.J., "Drift", Journal of Fluid Mechanics, Vol. 1, 1956, p. 31-53.

[15] Atassi, H.M. "The Sears Problem for a Lifting Airfoil Revisited - New Results", Journal of Fluid Mechanics, Vol. 141, 1984, p. 109-122.

[16] Goldstein, M.E., "Unsteady Vortical and Entropic Distortions of Potential Flows Round Arbitrary Obstalces", Journal of Fluid Mechanics, Vol. 89, 1978, p.433-468.

APPENDIX A

FORTRAN - 77 SOURCE CODE

```
c       *** Include file to define grid sizes ***
        integer NMAX, MMAX
c       ***     Name            Definition
c               NMAX            Max # of points in the axial mesh
c               MMAX            Max # of points in the radial mesh
        parameter(NMAX=100,MMAX=10)
```

```
      program gmain
c     *** main program for the calculation ***
c     *** program parameters:
c         eta     n vector of the axial coordinates on the mesh
c         rvec    m vector of the radial coordinates on the mesh
c         g       n x m array
c         h       n vector of the differences of the axial coords
c         xk      n x m array of spline parameters at each j
c         q       n x m array of forcing function on the mesh
c         alp     m x 2 array of tail rates of decay for each j
c         cf      m x 2 array of tail coefficients for each j
c         qs      n vector for scratch purposes
c         xks     n vector for scratch purposes
c         al      2 vector containing rate of decay for tail
c         cs      2 vector containing the tail coefficients
c         fint    n x m x 4 x 2 array containing the predigested
c                 integrals of the Green's function and the near
c                 field approximation (Done only once!)
c
      include "const.f"
      dimension eta(NMAX),rvec(MMAX),g(NMAX,MMAX),h(NMAX),xk(NMAX,MMAX)
      dimension q(NMAX,MMAX)
      dimension alp(MMAX,2),cf(MMAX,2),qs(NMAX),xks(NMAX),al(2),cs(2)
      dimension fint(NMAX,MMAX,4,2)
      real h
      integer har, kflag, hmax, cflag
      data n,m,xh/20,1,0.5/
      write(6,60)
   60 format(5x,'what are the number of intervals?')
      read(5,*) n
c     *** initialize the mesh ***
      call imesh(n,m,nout,eta,h,rvec)
c     *** calculate zeroth order approximation
      write(6,70)
   70 format(5x,'what is the maximum harmonic?')
      read(5,*) hmax
c     *** find the coefficients in the approximation ***
      n2 = n-2
      kflag = 0
c     *** set the flag to calculate the near field terms ***
      cflag = 0
c     *** compute the coefficients of the forcing term ***
      call cfor(n,m,eta,rvec,q,hmax)
c     *** isolate the harmonics and load data ***
      har=0
c     *** master loop for iteration ***
      do 800 k=1,5
        if ( k .gt. 1 ) kflag = 1
c       *** step through the harmonics ***
c       do 800 har = -hmax,hmax,1
c       *** run through the radial direction ***
          do 700 j=1,m
c         *** cover the mesh ***
            do 550 i=1,n
c             *** insert the values into the vector ***
```

```
                qs(i) = q(i,j)
  550           continue
c              write(6,450) (qs(jt),jt=1,n)
  450           format(/11e11.4)
c               *** calculate the k's for the spline approximation ***
                call ksolv(n,nout,eta,h,qs,xks,al,cs,kflag)
c               *** store the k's that result ***
                do 600 i=1,n
  600             xk(i,j) = xks(i)
c               *** store the constants for the decaying term ***
                do 625 i=1,2
                  alp(j,i) = al(i)
  625             cf(j,i) = cs(i)
                har = 0
                if( cflag .ne. 0) goto 650
                call fortrm(har,n,m,nout,eta,rvec,h,xk,fint)
c               *** mark the fact that the values are known ***
                cflag = 1
  650           continue
c             *** calculate the integral using the spline approximation ***
              call eval(har,n,m,nout,eta,rvec,g,h,xk,fint,q,alp,cf)
c             *** close the loop and test for convergence ***
c             *** print out the results ***
              call test2(n,m,eta,rvec,q,xk,g,alp,cf)
c             *** use calculated values for next pass ***
              do 675 i=1,n
                q(i,j) = g(i,j)
  675           continue
  700         continue
  800       continue
c         *** end of main iteration loop ***
          stop
          end
c
          subroutine imesh(n,m,nout,eta,h,rvec)
c         *** subroutine to interactively initialize the mesh ***
c         *** subroutine parameters:
c             n     number of axial points on the mesh
c             m     number of radial points on the mesh
c             nout  number of outlying points on the mesh
c             eta   n vector of axial coordinate points on the mesh
c             h     n vector of the first differences of the axial coords
c             rvec  m vector of the radial points on the mesh
c
          include "const.f"
          dimension eta(NMAX),h(NMAX),rvec(MMAX)
          real h
          write(6,100)
  100     format(5x,'what is the maximum extent of the mesh?')
          read(5,*) x
          sc = float(n/2)
          write(6,130)
  130     format(5x,'how many points outside the mesh?')
          read(5,*) nout
c         ***take care of the outer region***
```

```
      nold = n + 2*nout + 2
c     ***add refined mesh near origin***
      n = nold + 4
      ht = x/sc
      eta(1) = -(x+ht*nout)
      eta(n) =  (x+ht*nout)
      do 150 i=1,((n/2)-1)
        ni =  n - i
c       ***cut down the mesh size near the origin***
        if(i.eq.((nold/2)-1)) ht = ht/2.
        if(i.eq.(nold/2)) ht = ht/2.
        h(i) = ht
        h(ni) = ht
        eta(i+1) = eta(i) + ht
        eta(ni) = eta(ni+1) - ht
  150   continue
c     write(6,120) (eta(i),i=1,n)
  120 format(5e15.6)
      rvec(1) = 1.0
      return
      end
c
c
      subroutine mtest(n,m,eta,rvec,q,xk,al,cs)
      include "const.f"
      dimension eta(NMAX),rvec(MMAX),q(NMAX),xk(NMAX)
      dimension al(MMAX,2),cs(MMAX,2)
      do 140 j=1,m
  110   format(5x,4f12.5)
        write(6,110) (al(j,k),k=1,2)
        write(6,110) (cs(j,k),k=1,2)
        write(6,120) eta(1),q(1)
  120   format(//10x,f7.2,5x,e12.5)
        do 140 i=2,n-1
          qp = 0.0
          if(abs(eta(i)).le. 1.e-3) goto 125
          if(eta(i).gt.0.) qp=al(j,2)*cs(j,2)/abs(eta(i))**(al(j,2)+1)
          if(eta(i).lt.0.) qp=al(j,1)*cs(j,1)/abs(eta(i))**(al(j,1)+1)
  125     write(6,130) eta(i),q(i),xk(i),qp
  130     format(10x,f7.2,5x,e12.5,5x,e12.5,5x,e12.5)
  140   continue
      write(6,150) eta(n),q(n)
  150 format(10x,f7.2,5x,e12.5)
      return
      end
c
c
      subroutine test2(n,m,eta,rvec,q,xk,g,al,cs)
      include "const.f"
      dimension eta(NMAX),rvec(MMAX),q(NMAX,MMAX),xk(NMAX,MMAX)
      dimension al(MMAX,2),cs(MMAX,2),g(NMAX,MMAX)
1     do 145 j=1,m
  110   format(5x,4f13.5)
        write(6,110) (al(j,k),k=1,2)
        write(6,110) (cs(j,k),k=1,2)
```

```
          write(6,120) eta(1),q(j,1)
  120     format(/2x,'test2:'/7x,'1',2(5x,e13.5))
          do 140 i=2,n-1
            write(6,130) i,eta(i),q(i,j),xk(i,j),g(i,j)
  130       format(5x,i3,4(5x,e13.5))
  140       continue
  145     write(6,150) n,eta(n),q(n,j)
  150   format(5x,i3,2(5x,e13.5))
        return
        end
c
        subroutine stest(n,m,nout,eta,h,xk,q,alp,cf,khop,d)
        include "const.f"
        dimension eta(NMAX),h(NMAX),xk(NMAX,MMAX),q(NMAX,MMAX)
        dimension alp(MMAX,2),cf(MMAX,2),d(3),val(NMAX)
        dimension al(2),cs(2),ap(NMAX),z(NMAX)
c       *** statement function for evaluating spline ***
        seval(t) = q(i+1,j)*t + q(i,j)*t1 + h(i)*((xk(i,j)-d1)*t1
     1             - (xk(i+1,j)-d1)*t)*t*t1
c       *** initialize the test statistics ***
        data kmax/10/
c       *** set up for right hand side of mesh ***
        k = n/2
        km1 = k - 1
        k1  = k + 1
c       *** set up the loops ***
        do 60 j = 1,m
          al(1) = alp(j,1)
          al(2) = alp(j,2)
          cs(1) = cf(j,1)
          cs(2) = cf(j,2)
          dmax = 0.0
          rmax = 0.0
          x12  = 0.0
c         *** go over the points in the refined mesh ***
          xkmax = float(kmax)
          kount = 0
          do 50 i = k1,(n-nout-1)
            d1 = (q(i+1,j)-q(i,j))/h(i)
            do 20 k = 1,kmax
c             *** evaluate on the sub-mesh points ***
              t = float(k)/xkmax
              t1 = 1.0 - t
              z(k)  = eta(i) + t*h(i)
              ap(k) = seval(t)
   20         continue
            call tail(n,eta,kmax,z,al,cs,khop,d,val)
            do 30 k=1,kmax
c             *** update the statistics ***
              dev = abs(ap(k) - val(k))
              rel = dev/abs(val(k))
              dmax = amax1(dmax,dev)
              rmax = amax1(rmax,rel)
              x12  = x12 + dev*dev
              kount = kount + 1
```

```
c             if((i-k1).le.3)write(6,90) kount,z(k),ap(k),val(k),dev,rel
   90         format(5x,i3,5e14.5)
   30         continue
   50       continue
          rms = sqrt(x12/float(kount))
          write(6,100) dmax,rmax,rms
  100     format(5x,'the maximum error is ',e15.5/5x,
     1    'the maxium relative error is ',e15.5/5x,'the rms is ',e15.5)
   60     continue
        return
        end
c
        subroutine tail(n,eta,m,z,al,cs,khop,d,val)
        include "const.f"
        dimension z(NMAX),eta(NMAX),val(NMAX),al(2),cs(2),d(3)
c       *** statement functions for the problem ***
        fun(x,a,c) = c*abs(x)**-a
        facp(a,y) = 1.+d(1)*a*y*(1.+d(2)*(a+1)*y*(1.+d(3)*(a+2.)
     1  *y/3.)/2.)
        facm(a,y) = 1.-d(1)*a*y*(1.-d(2)*(a+1.)*y*(1.-d(3)*(a+2.)
     1  *y/3.)/2.)
        n2 = n/2
c       *** evaluate the function on the given points ***
        do 100 i=1,m
        x = z(i)
c       *** which side of the mesh is the point on? ***
        if( x .gt. 0 ) goto 50
c       *** do calculation for negative side ***
        c = cs(1)
        a = al(1)
c       *** calculate point where matching is done ***
        estar = eta(n2 - khop)
c       *** which formula is used? ***
        if( x-estar .lt. -0.0005 ) goto 25
        y = (x/estar) - 1.
        val(i) = fun(estar,a,c)*facm(a,y)
        goto 100
   25   val(i) = fun(x,a,c)
        goto 100
c       *** do calculation on positive side ***
  50    c = cs(2)
        a = al(2)
        estar = eta(n2+1+khop)
c       *** which formula is used? ***
        if( x-estar .gt. 0.0005) goto 75
        y = (x/estar) - 1.
        val(i) = fun(estar,a,c)*facm(a,y)
        goto 100
  75    val(i) = fun(x,a,c)
  100   continue
c       if( n .eq. m )write(6,110) (val(i),i=1,m)
  110   format(5x,5e15.5)
        return
        end
```

```
      subroutine cfor(n,m,eta,rvec,q,hmax)
c     *** subroutine parameters:
c         n   size of eta (axial direction) vector
c         m   size of rvec (radial direction) vector
c         eta vector containing axial coordinates for grid
c         rvec vector containing radial coordinates for grid
c         q   nxm array of the forcing function
c         har number of harmonic being used
c
      include "const.f"
      dimension eta(NMAX),rvec(MMAX),q(NMAX,MMAX),qs(MMAX),xk(MMAX)
      dimension xh(MMAX),d(MMAX)
c     ***this code forms the forcing function***
c     ***(only radial term is currently here)***
      do 30 i=1,n
        do 10 k=1,m
          qs(k) = q(i,k)
   10     continue
c       ***solve for the spline coefficients***
        call ksolv2(m,rvec,qs,xh,d,xk)
        do 20 k=1,(m-1)
          r = rvec(k)
          if (k.eq.1) then r = amin1(abs(rvec(1)), .00001)
c         ***calculate the radial portion of the forcing term***
          q(i,k) = xk(k)/r + 2.*(3.*d(k) - 2.*xk(k) - xk(k+1))/xh(k)
   20     continue
c       ***expression for the last point on the mesh***
        r = rvec(m)
        m1 = m -1
        q(i,m) = xk(m)/r - 2.*(3.*d(m1) -xk(m1) - 2.*xk(m))
     1    /xh(m1)
c       ***repeat for the next mesh column***
   30   continue
      return
      end
c
      subroutine fortrm(har,n,m,nout,eta,rvec,h,xk,fint)
c     *** subroutine to do the predigested portions of the integral***
c     *** resulting from integrating the Green's function against ***
c     *** the spline approximation in the near field and the tail ***
c     *** in the far field ***
c     *** subroutine parameters:
c         har    number of the harmonic being considered
c         n      number of axial coordinates on the grid
c         m      number of radial coordinates on the grid
c         nout   number of 'out lying' points to estimate the tail
c         eta    n vector of the axial coordinates on the grid
c         rvec   m vector of the radial coordinates on the grid
c         h      n vector of the axial differences on the grid
c         xk     n x m array of spline parameters
c         fint   n x m x 4 x 2 results from evaluating the near
c                field subintegrals
c
c     *** this subroutine only needs to be called once, the ***
c     *** results are the same for each pass and harmonic ***
```

```
c
      include "const.f"
      dimension eta(NMAX),rvec(MMAX),frvec(MMAX),h(NMAX)
      dimension xk(NMAX,MMAX),fint(NMAX,MMAX,4,2)
      integer har
c     ***here follows the statement functions for calculating the
c        Green's function using a piecewise cubic polynomial
c        approximation between the nodes in the mesh along the
c        lateral direction. ***
c     ***functions for recursion relationships***
      g00(a,b)=(fc-fa)*c2
      g10(a,b)=(c3-f00)*c2
      g01(a,b)=(f00-c4)*c2
      g20(a,b)=(fc-2.*f10)*c2
      g11(a,b)=(f10-f01)*c2
      g02(a,b)=(2.*f01-fa)*c2
      g21(a,b)=(f20-2.*f11)*c2
      g12(a,b)=(2.*f11-f02)*c2
c     ***note that in the expressions above that several quantities
c        must be defined in the program before the statement function
c        will make sense.  the explanation for doing this is that it
c        saves the overhead of mxn function calls needed to evaluate
c        the integrals involved.  this could save a lot of time.
c     ***statement functions for the problem ***
      f(r) = sqrt(v0(r) + (float(har)/r)**2)
c     *** with the preliminary definitions done, it is now time
c         to set up the calculation.  first, find k ***
      k = n/2
      if(n .eq. 2*k) go to 10
c     *** write error message and halt if n is odd ***
      write(6,100) n
 100  format(5x,'n = ',i5,' and must be even. program stopped.')
      stop
c     *** check to see if both 0+ and 0- are as expected***
   10 k1 = k + 1
      km1 = k - 1
      if(abs(eta(k1)+eta(k)).le. 0.005) go to 20
      write(6,110) eta(k),eta(k1)
 110  format(5x,'program halted.  representations of 0 are ',2e15.7)
      stop
c     *** set up the fr vector for the calculations ***
   20 do 22 j=1,m
      r = rvec(j)
      frvec(j) = f(r)
      if( abs(frvec(j)) .lt. 0.00005) goto 23
   22 continue
      goto 24
   23 write(6,120) j,fr
  120 format(5x,'on line ',i3,' the fr value ',e12.5,' is too small')
      stop
c     *** enter main loop to calculate values on the mesh ***
c     *** set up for the radial sweep ***
   24 do 95 j=1,m
      fr = frvec(j)
c     *** code for calculating downstream of actuator ***
```

```
      do 56 kl=(nout+1),km1
c     ***calculate the constants first***
      a = eta(kl) * fr
      b = h(kl) * fr
      c = a + b
      c2 = 1./b
c     ***first case***
      fc = sinh(c)
      fa = sinh(a)
      c3 = cosh(c)
      c4 = cosh(a)
      jmp = 1
c     ***code for calculating subintegrals by recursion***
 40   f00 = g00(a,b)
      f10 = g10(a,b)
      f01 = g01(a,b)
      f20 = g20(a,b)
      f11 = g11(a,b)
      f02 = g02(a,b)
      f21 = g21(a,b)
      f12 = g12(a,b)
c     write(6,130) kl,a,b,c,c2,fa,fc,c3,c4,f00,f10,f01
 130  format(i3,5x,4f7.2,5x,4e14.5,5x,3e14.5)
c     ***jump to code for evaluating the contribution***
      goto (45,55,75,85),jmp
 45   fint(kl,j,1,1) = f10
      fint(kl,j,2,1) = f01
      fint(kl,j,3,1) = f12
      fint(kl,j,4,1) = f21
c     ***second case***
 50   fc = exp(c)
      fa = exp(a)
      c3 = fc
      c4 = fa
      b  = -1.*b
      jmp = 2
c     ***calculate the recursion results***
      goto 40
c     ***evaluate the contribution***
 55   fint(kl,j,1,2) = f10
      fint(kl,j,2,2) = f01
      fint(kl,j,3,2) = f12
      fint(kl,j,4,2) = f21
 56   continue
c
c     *** this code is for upstream of the actuator ***
 60   do 90 kl=kl,(n-(nout+1))
c     *** repeat the process on the other side of the mesh ***
      a = eta(kl)*fr
      b = h(kl)*fr
      c = a+b
      b = -1.*b
      c2 = 1./b
c     ***first case***
      fa = exp(-a)
```

```
      fc = exp(-c)
      c3 = fc
      c4 = fa
      jmp = 3
c     ***evaluate recursion formula***
      goto 40
c     ***unscramble the results of the recursion***
c     ***now, carry out the evaluation***
   75 fint(kl,j,1,1) = f10
      fint(kl,j,2,1) = f01
      fint(kl,j,3,1) = f12
      fint(kl,j,4,1) = f21
c     ***second case ***
   80 fc = sinh(c)
      fa = sinh(a)
      c3 = cosh(c)
      c4 = cosh(a)
      b = -1.*b
      c2 = 1./b
      jmp = 4
c     ***evaluate recursion formula***
      goto 40
   85 fint(kl,j,1,2) = f10
      fint(kl,j,2,2) = f01
      fint(kl,j,3,2) = f12
      fint(kl,j,4,2) = f21
   90 continue
   95 continue
c     ***done!***
      return
      end
c
      subroutine ksolv(n,nout,x,h,q,k,as,cs,kflag)
c     *** this subroutine finds the spline coefficients in the axial
c     *** direction given the values ***
c     *** subroutine parameters:
c         n     number of axial coordinates in the mesh
c         nout  number of 'out lying' points to estimate the decay
c         x     n vector of axial coordinates on the mesh
c         q     n vector of the function values on the mesh
c         k     n vector of the spline parameters (output)
c         as    2 vector of the decay rate of the tail
c         cs    2 vector of the coefficient for the tail
c         kflag flag to indicate if tail estimates exist
c
      include "const.f"
      dimension q(NMAX),x(NMAX),k(NMAX),h(NMAX),as(2),cs(2)
      dimension cm(NMAX,4),cp(NMAX,4),km(NMAX),kp(NMAX)
      real h,k0,kn2,k,km,kp
      integer kflag
c     *** statement functions for this routine ***
      al(xv,y,u,v) = alog(v/y)/alog(abs(xv/u))
      ce(f,x,a) = f*abs(x)**a
c
c     write(6,120) (q(i),i=1,n)
```

```
 120   format(/11e11.4)
       if( kflag .ne. 0) goto 15
       xl = x(2)
       yh = x(n-1)
       fl = q(2)
       gh = q(n-1)
       if( q(1) * fl .le. 0) goto 40
       if( gh * q(n) .le. 0) goto 50
       alp1 = al(x(1),q(1),xl,fl)
       alp2 = al(yh,gh,x(n),q(n))
       do 10 i=2,(nout+2)
         xh = x(i+1)
         fh = q(i+1)
         yl = x(n-i)
         gl = q(n-i)
         if( fl*fh .le. 0 ) goto 40
         if( gl*gh .le. 0 ) goto 50
         alp1 = alp1 + al(xl,fl,xh,fh)
         alp2 = alp2 + al(yl,gl,yh,gh)
         xl = xh
         fl = fh
         yh = yl
         gh = gl
  10     continue
       xkt = float(nout+2)
       as(1) = alp1/xkt
       as(2) = alp2/xkt
  15   c1 = 0
       c2 = 0
       alp1 = as(1)
       alp2 = as(2)
       do 20 i=1,(nout+2)
         c1 = c1 + ce(q(i),x(i),alp1)
         ni = n - i + 1
         c2 = c2 + ce(q(ni),x(ni),alp2)
  20     continue
       c = c1/xkt
       cd = c2/xkt
       cs(1) = c
       cs(2) = cd
       fac = c*alp1
       fac2 = -cd*alp2
       k0 = fac/abs(x(nout+1))**(alp1+1)
       k(nout+1) =k0
       kn2 = fac2/x(n-nout)**(alp2+1)
c      ***calculate the interpolated value***
c      ***3 point Lagrangian interpolation scheme***
       n2 = n/2
       t = h(n2-1) + h(n2-2)
       ym = 0.5*((h(n2-2)+0.5*h(n2-1))*((q(n2-1)/h(n2-1))+(q(n2)/
     1 t))-0.5*q(n2-2)*(h(n2-1)*h(n2-1))/(h(n2-2)*t))
       print *,'ksolv: t',t,' ym',ym
       n2p = n2 + 1
       t = h(n2p) + h(n2p+1)
       yp = 0.5*((h(n2p)+0.5*h(n2p+1))*((q(n2p)/t)+(q(n2p+1)/
```

```
     1  h(n2p+1)))-0.5*q(n2p+2)*(h(n2p)*h(n2p))/(h(n2p+1)*t))
        print *,'ksolv: t',t,' yp',yp
c       ***set up the coefficient matrices***
        ncoef = n2 - nout
c       ***     first row       ***
        cm(1,1) = 1.
        cm(1,2) = 0.
        cm(1,3) = 0.
        cm(1,4) = k0
c       ***     last row        ***
        cm(ncoef,1) =  0.
        cm(ncoef,2) = -1.
        cm(ncoef,3) =  1.
        cm(ncoef,4) = (q(n2) + q(n2-1) - 2.*ym)*(4./h(n2-1))
c       ***     interior        ***
        d0 = (q(nout+2) - q(nout+1))/h(nout+1)
c       print *,'ksolv: i    1c:',cm(1,1),cm(1,2),cm(1,3),cm(1,4)
        do 25 i=2,ncoef-1
          j = i + nout
          d1 = (q(j+1)-q(j))/h(j)
          cm(i,1) = h(j)
          cm(i,2) = 2.*(h(j)+h(j-1))
          cm(i,3) = h(j-1)
          cm(i,4) = 3.*(h(j-1)*d1 + h(j)*d0)
          d0 = d1
c         print *,'ksolv: i ',i,'cm:',cm(i,1),cm(i,2),cm(i,3),cm(i,4)
   25     continue
c       print *,'ksolv: i ',ncoef,'cm:',cm(ncoef,1),cm(ncoef,2),
c     1 cm(ncoef,3),cm(ncoef,4)
c       ***     last row        ***
        cp(ncoef,1) = 0.
        cp(ncoef,2) = 1.
        cp(ncoef,3) = -1.
        cp(ncoef,4) = (q(n2p) + q(n2p+1) - 2.*yp)*(4./h(n2p))
c       ***     first row       ***
        cp(1,1) = 1.
        cp(1,2) = 0.
        cp(1,3) = 0.
        cp(1,4) = kn2
c       ***     interior        ***
        d0 = (q(n2p+1)-q(n2p))/h(n2p)
        do 30 i=2,ncoef-1
          j = n2p + i - 1
c         ***     switch order    ***
          d1 = (q(j+1)-q(j))/h(j)
          cp(ncoef+1-i,1) = h(j-1)
          cp(ncoef+1-i,2) = 2.*(h(j)+h(j-1))
          cp(ncoef+1-i,3) = h(j)
          cp(ncoef+1-i,4) = 3.*(h(j-1)*d1 + h(j)*d0)
          d0 = d1
   30     continue
c       do 31 i=1,ncoef
c         print *,'ksolv:',i,' cp: ',cp(i,1),cp(i,2),cp(i,3),cp(i,4)
   31     continue
c       *** call the tri-diagonal solver routine ***
```

```
      call trid(cm,km,ncoef)
      call trid(cp,kp,ncoef)
c     *** insert the returned values ***
      do 35 i =1,ncoef
        k(i+nout) = km(i)
        k(n2p+ncoef-i) = kp(i)
        print *,'ksolv: ',i,km(i),kp(i)
 35     continue
      return
 40   write(6,100)
100   format(5x,'values on the negative mesh change sign')
      stop
 50   write(6,110)
110   format(5x,'values on the positive mesh change sign')
      stop
      end
c
      function v0(r)
c     *** the function subroutine supplies the mean harmonic ***
c     *** for the flow into the actuator ***
c     *** this is just a test function and not the real thing ***
      v0 = 1.0
      return
      end
c
      subroutine eval(har,n,m,nout,eta,rvec,g,h,xk,fint,q,alp,cf)
c     *** subroutine to evaluate the integral of the Green's ***
c     *** function both in the near and far field ***
c     *** subroutine parameters:
c         har    number of the harmonic being considered
c         n      number of axial coordinates
c         m      number of radial coordinates
c         nout   number
c         eta    n vector containing the axial coordinates
c         rvec   m vector containg the radial coordinates
c         g      n x m array containing integration results
c         h      n vector containing the difference in axial coords
c         xk     n x m array containing the spline parameters
c         fint   n x m x 4 x 2 array containing pre-digested integrals
c                resulting from the spline approximation
c         q      n x m array containing the forcing function
c         alp    2 vector containing the estimate of the rate of decay
c                both before and after the actuator
c         cf     2 vector containing the estimate of the coefficient
c                for the tail of the decay
c
      dimension eta(NMAX),rvec(MMAX),g(NMAX,MMAX),h(NMAX)
      dimension xk(NMAX,MMAX),q(NMAX,MMAX),alp(MMAX,2),cf(MMAX,2)
      dimension fint(NMAX,MMAX,4,2)
      integer har
c     ***statement functions for the problem ***
      f(r) = sqrt(v0(r) + (float(har)/r)**2)
      fsum(kl,a,b)=q(kl+1,j)*fint(kl,j,1,kt) + q(kl,j)*
     1 fint(kl,j,2,kt) + h(kl)*((xk(kl,j)-d)*fint(kl,j,3,kt) -
     2 (xk(kl+1,j)-d)*fint(kl,j,4,kt))
```

```
      ft1(x,ia,f1) = ei(ia,f1,x)
      ft2(x,ia,f1) = sinhi(ia,del,z1)/(xmesh**(ia-1))
c
c     ***set up needed constant parameters***
      k = n/2
      km1 = k - 1
      k1 = k + 1
      b = 0.0
      xmesh = abs(eta(nout+1))
c     write(6,110) alp(1,1),alp(1,2),cf(1,1),cf(1,2)
  110 format(5x,4e15.5)
      do 95 j=1,m
        fr = f( rvec(j) )
        z1 = xmesh*fr
c       *** enter main loop to calculate values on the mesh ***
   24   do 95 i=1,n
c         ***set the axial coordinate***
          x = eta(i)
          c1 = x*fr
c         *** determine the side of the actuator used ***
   25     if(i .ge. k1) go to 60
c         *** this code is for upstream of the actuator ***
c         ***calculate the far field contribution***
          a = alp(j,1)
          c = a + fr
          ia = int(a+0.5)
          ax = abs(x)
          if( i .gt. nout ) goto 30
          s1 = cf(j,1)*ft1(ax,ia,fr)
          del = (ax - xmesh)/xmesh
          s2 = cf(j,1)* (-1.)*ft2(ax,ia,fr)
c         ***compute and store the far field for the mesh points***
          ft1x = ft1(xmesh,ia,fr)
          goto 35
   30     s1 = cf(j,1)*ft1x
          s2 = 0.0
   35     continue
c         *** sum over the intervals in the mesh***
          do 57 kl = (nout+1),km1
            klp = kl
            d = (q(kl+1,j)-q(kl,j))/h(kl)
c           *** is the field point to the right of the mesh point? ***
            if( i .gt. kl) goto 50
c           ***first case***
            kt = 1
   45       s2 = s2 + h(kl)*fsum(klp,a,b)
            goto 56
c           ***second case ***
   50       kt = 2
            s1 = s1 + h(kl)*fsum(klp,a,b)
   56       continue
c           if(i .ne. 1) goto 57
c           write(6,100) i,kl,kt,s1,s2,d,(fint(kl,j,lt,kt),lt=1,4)
  100       format(5x,i3,2i5,3e15.5,5x,4e15.5)
   57       continue
```

```
            goto 92
c
c           ***calculate the far field***
   60       a = alp(j,2)
            c = a + fr
            ia = int(a+0.5)
            if( i .gt. (n-nout)) goto 65
            s1 = -cf(j,2)*ft1x
            s2 = 0.0
            goto 70
   65       s1 = -cf(j,2)*ft1(x,ia,fr)
            del = (x - xmesh)/xmesh
            s2 = -cf(j,2)*ft2(x,ia,fr)
   70       do 90 kl=k1,(n-(nout+1))
              klp = kl
              d = (q(kl+1,j)-q(kl,j))/h(kl)
c             *** is the field point to the left of the mesh point? ***
              if( kl .lt. i ) go to 80
c             ***first case***
              kt = 1
              s1 = s1 - h(kl)*fsum(klp,-c,fr)
              goto 89
c             ***second case***
   80         kt = 2
              s2 = s2 - h(kl)*fsum(klp,a,b)
   89         continue
c             if( i .ne. n) goto 90
c             write(6,100) i,kl,kt,s1,s2,d,(fint(kl,j,lt,kt),lt=1,4)
   90       continue
   92       g(i,j) = s1*sinh(c1) + s2*exp(-abs(c1))
   95     continue
        return
        end
c
        function ei(ia,f,x)
c       *** subroutine for the evaluation of the second integral ***
c       *** resulting from integrating the Green's function ***
c
c       ***test for special case when arguement is zero***
        if( abs( f*x ) .ge. 1.e-05) goto 5
        ei = 1./float(ia - 1)
        return
c       ***initialize starting values***
    5   fs = 0.
        fm = -1. * f
        fx = fm * x
        c = 1.0
        if ( ia .eq. 1 ) goto 20
c       ***do integration by parts until e/sub 1/ is reached***
        do 10 i=ia,2,-1
          i1 = i - 1
          p = float(i1)
          fs = fs + (c/p)*exp(fx)/x**i1
          c = c * fm/p
   10     continue
```

```
c       ***calculate e/sub 1/ using continued fraction approximation
c          from page 231 of abramowitz & stegun***
   20   fx = -1. * fx
        e1 = 1.039549/(1.43679 + fx)
        e1 = 0.99592/(1.893888 + fx - e1)
        e1 = (1. - e1)*exp(-fx)/fx
c       ***final step***
        fs = fs + c*e1
        ei = fs
        return
        end
c
        subroutine romb(funct,xlow,xup,tol,val,xm)
c       *** this subroutine calculates a definite integral using the ***
c       *** extrapolation scheme based on the trapezoidal rule ***
        dimension x(256),fval(256),y(10,10)
        external funct
        do 10 i=1,10
          do 10 j=1,10
 10         y(i,j) = 0.0
        sum = 0.0
        x(1) = xlow
        x(2) = (xlow+xup)/2.
        x(3) = xup
        call funct(x,3,fval,xm)
        sum = sum+fval(1)+fval(3)+2*fval(2)
        step = (xup-xlow)/2.
        y(1,1)=.5*step*sum
        do 50 i=1,7
          step = step/2.
          do 20 j=1,2**i
 20         x(j) = xlow + (-1+2*j)*step
          call funct(x,2**i,fval,xm)
          temp=0.0
          do 30 j=1,2**i
 30         temp=temp + fval(j)
          sum = sum + 2*temp
          y(1,i+1)=.5*step*sum
          n = 0
          do 40 j=2,i+1
            n = 2+n
 40         y(j,i+1)=((y(j-1,i+1)*2**n)-y(j-1,i))/float(-1+2**n)
          if (abs(y(i+1,i+1)-y(i,i+1)).le. (tol*y(i,i+1)))
     #      goto 60
 50       continue
c       write(6,100)
100     format(5x,'warning, romberg failed; last value used')
        i=i-1
 60     val=y(i+1,i+1)
        return
        end
c
        subroutine func(x,n,fval,xm)
c       *** function to calculate the cosh term ***
        dimension x(256),fval(256)
```

```
      do 10 i=1,n
        z = x(i)
        fval(i) = cosh(z*xm)/z
   10   continue
      return
      end
c
      subroutine funs(x,n,fval,xm)
c     *** function to calculate the sinh term ***
      dimension x(256),fval(256)
      do 10 i=1,n
        z = x(i)
        fval(i) = sinh(z*xm)/z
   10   continue
      return
      end
c
      function sinhi(ia,d,xnu)
c     *** subroutine to calculate the Sinhi integral for the ***
c     *** integration of the Green's Function ***
      external func,funs
c     ***initialize starting values***
      f = 0.
      c = 1.
      iflag = 1
      x = 1. + d
      if( x*xnu .ge. 85.) write(6,100) ia,d,xnu
  100 format(5x,'ERROR--overflow for combination ',i5,2e15.7)
      cu = cosh(x*xnu)
      cl = cosh(xnu)
      su = sinh(x*xnu)
      sl = sinh(xnu)
      if( ia .eq. 1 ) goto 32
c     ***do the integration by parts until base value is reached***
      do 30 i=ia,2,-1
        i1 = i - 1
        p =float(i1)
        if(iflag .eq. 0) goto 15
        fu = su/x**i1
        fl = sl
        iflag = 0
        goto 20
   15   fu = cu/x**i1
        fl = cl
        iflag = 1
   20   f = f + (c/p)*(fl - fu)
        c = c * (xnu/p)
   30   continue
c     insert code here for shi(x)-shi(1.) calculation
   32 xone = 1.0
      tol = 1.e-04
c     current code uses romberg extrapolation for integral
      if(iflag .eq. 0) goto 35
      call romb(funs,xone,x,tol,val,xnu)
      f = f + c*val
```

```
      goto 40
   35 call romb(func,xone,x,tol,val,xnu)
      f = f + c*val
   40 continue
      sinhi = f
      return
      end
c
c     ***(current end of program)***
```

```
      subroutine ksolv2(m,rvec,qs,xh,d,xk)
c     *** subroutine to calculate the spline approximation in the ***
c     *** radial direction ***
c     *** subroutine parameters:
c         m     number of radial points on the mesh
c         rvec  m vector of the radial coordinates of the mesh
c         qs    m vector of the functional values
c         xh    m vector of differences of the radial coordinates
c         d     m vector of the finite differences of the function
c         xk    m vector of the spline parameters for the function
c
      include "const.f"
      dimension rvec(20),xh(20),d(20),xk(NMAX),qs(20),c(MMAX,4)
c     ***set up the constants for the spline calculation ***
      print *,'ksolv2: i, x, xh, d'
      do 10 i=1,(m-1)
        xh(i) = rvec(i+1) - rvec(i)
        d(i) = (qs(i+1) - qs(i))/xh(i)
        print *,i,rvec(i),xh(i),d(i)
   10   continue
c
c     ***set the alternative boundary condition***
      t = (xh(1) + xh(2))
      y1 = ((xh(2)+t)*((qs(1)/t)+(qs(2)/xh(2)))-qs(3)*
     &     xh(1)*xh(1)/(xh(2)*t))*0.25
      t = (xh(m-1)+xh(m-2))
      yn = ((t+xh(m-2))*((qs(m-1)/xh(m-2))+(qs(m)/t))
     &     - qs(m-2)*xh(m-1)*xh(m-1)/(xh(m-2)*t))*0.25
c     ***y1 and y2 are estimates of the mid-point of the ends***
c
      print *,'ksolv2: y1=',y1,' yn = ',yn
c     ***set first row of tri-diagonal matrix system***
c     ***use alternative condition***
      c(1,1) = -1 .0
      c(1,2) =  1.0
      c(1,3) =  0.0
      c(1,4) = (qs(1) + qs(2) - 2.*y1)*(4./xh(1))
c     ***add the far edge condition to the system***
c     ***use alternative condition***
      c(m,1) =  0.0
      c(m,2) = -1.0
      c(m,3) =  1.0
      c(m,4) = (qs(m) + qs(m-1) - 2.*yn)*(4./xh(m-1))
      d1 = xh(1)
c     ***fill in interior rows of matrix system***
      do 20 i=2,(m-1)
        d2 = xh(i)
        c(i,1) = d2
        c(i,2) = 2.*(d1+d2)
        c(i,3) = d1
        c(i,4) = 3.*(d1*d(i) + d2*d(i-1))
        d1 = d2
   20   continue
c     ***call the tri-diagonal matrix solver***
      print *,'ksolv2: i, c(i,1) c(i,2), c(i,3) c(i,4)'
```

```
      do 30 i=1,m
        write(6,100) i,(c(i,j),j=1,4)
 30     continue
100   format(i3,4e15.5)
      write(6,110)
110   format(/)
      call trid(c,xk,m)
      print *,'ksolv2: i, c(i,1) c(i,2) c(i,3) c(i,4)'
      do 40 i=1,m
        write(6,100) i,(c(i,j),j=1,4)
 40     continue
      print *, 'problem spot in code'
      return
c     print *, 'will this do the trick?'
      end
c
      subroutine trid( c, f, n )
c     *** solving  a set of n linear simultaneous equations
c         having a tridiagonal coefficient matrix.
c         coefficient matrices are represented by c(i,j) and
c         the solution is stored in the one-dimensional
c         array f(i) ***
      include "const.f"
      dimension c(NMAX,4),f(NMAX)
c     *** eliminate th first term of each of the equations
c         (except the first and last one) according
c         to the gaussian scheme ***
      nm1 = n-1
c     *** do the elimination on the second row ***
      c(2,2) = c(1,1)*c(2,2) - c(2,1)*c(1,2)
      c(2,3) = c(1,1)*c(2,3) - c(2,1)*c(1,3)
      c(2,4) = c(1,1)*c(2,4) - c(2,1)*c(1,4)
c     *** loop through the interior rows ***
      do 1 i=3,nm1
        c(i,2) = c(i,2)*c(i-1,2) - c(i,1)*c(i-1,3)
        c(i,3) = c(i,3)*c(i-1,2)
 1      c(i,4) = c(i,4)*c(i-1,2) - c(i,1)*c(i-1,4)
c     *** check and eliminate first element of last row ***
      if( c(n,1) .eq. 0. ) goto 2
c     *** eliminate the first element ***
      c(n,2) = c(n-2,1)*c(n,2) - c(n,1)*c(n-2,1)
      c(n,3) = c(n-2,1)*c(n,3)
      c(n,4) = c(n-2,1)*c(n,4) - c(n,1)*c(n-2,4)
 2    continue
c     *** find the first element by solving the 2x2 ***
      f(n) = ( c(n-1,2)*c(n,4)-c(n,2)*c(n-1,4) ) /
     1       ( c(n-1,2)*c(n,3)-c(n-1,3)*c(n,2) )
c     *** compute the solution using back substitution ***
      do 3 k=1,(nm1-1)
        j = n-k
 3      f(j) = ( c(j,4)-c(j,3)*f(j+1) ) / c(j,2)
c     *** solve for the first element ***
      f(1) = (c(1,4) - c(1,3)*f(3) - c(1,2)*f(2))' '1,1)
      return
      end
```

APPENDIX B

FORTRAN - 77 TEST CODE

```
      PROGRAM GHRMN
      DIMENSION ETA(100),RVEC(10),G(100,10),H(100),XK(100,10),Q(100,10)
      DIMENSION ALP(10,2),CF(10,2),QS(100),XKS(100),AL(2),CS(2)
      DIMENSION FINT(100,10,4,2),D(3)
      REAL H
      INTEGER HAR, KFLAG
      DATA N,M,XH/20,1,0.5/
      WRITE(6,60)
   60 FORMAT(5X,'WHAT ARE THE NUMBER OF INTERVALS?')
      READ(5,70) N
   70 FORMAT(I)
C     *** INITIALIZE THE MESH ***
      CALL IMESH(N,M,NOUT,ETA,H,RVEC)
C     *** CALCULATE ZEROTH ORDER APPROXIMATION
      WRITE(6,65)
   65 FORMAT(5X,'ENTER 1 FOR TAIL TEST, 2 FOR POLYNOMINAL TEST')
      READ(5,70) NC
      IF( NC .NE. 1 ) GOTO 50
      CALL PZERO(N,M,ETA,RVEC,Q,KHOP,D)
      GOTO 90
   50 CALL PONE(N,M,ETA,RVEC,Q,XK,AL,CS)
   90 CONTINUE
C     *** FIND THE COEFFICIENTS IN THE APPROXIMATION ***
      N2 = N-2
      KFLAG = 0
C     STOP
C     *** COMPUTE THE COEFFICIENTS OF THE FORCING TERM ***
C     CALL CFOR()
C     *** ISOLATE THE HARMONICS AND LOAD DATA ***
      HAR=0
C     *** MASTER LOOP FOR ITERATION ***
      DO 800 K=1,5
C     *** STEP THROUGH THE HARMONICS ***
      DO 700 J=1,M
C     *** COVER THE MESH ***
      DO 550 I=1,N
C     *** INSERT THE VALUES INTO THE VECTOR ***
      QS(I) = Q(I,J)
  550 CONTINUE
C     WRITE(6,450) (QS(JT),JT=1,N)
  450 FORMAT(/11E11.4)
C     *** CALCULATE THE K'S FOR THE SPLINE APPROXIMATION ***
      CALL KSOLV(N,NOUT,ETA,H,QS,XKS,AL,CS,KFLAG)
      KFLAG = 1
C     *** STORE THE K'S THAT RESULT ***
      DO 600 I=1,N
  600 XK(I,J) = XKS(I)
C     *** STORE THE CONSTANTS FOR THE DECAYING TERM ***
      DO 625 I=1,2
      ALP(J,I) = AL(I)
  625 CF(J,I) = CS(I)
      IF( NC .EQ. 1) CALL STEST(N,M,NOUT,ETA,H,XK,Q,ALP,CF,KHOP,D)
      HAR = 0
      IF( K .GE. 2) GOTO 650
      CALL FORTRM(HAR,N,M,NOUT,ETA,RVEC,H,XK,FINT)
```

```
  650   CONTINUE
C       *** CALCULATE THE INTEGRAL USING THE SPLINE APPROXIMATION ***
        CALL EVAL(HAR,N,M,NOUT,ETA,RVEC,G,H,XK,FINT,Q,ALP,CF)
C       *** CLOSE THE LOOP AND TEST FOR CONVERGENCE ***
C       *** PRINT OUT THE RESULTS ***
        CALL TEST2(N,M,ETA,RVEC,Q,XK,G,ALP,CF)
C       *** USE CALCULATED VALUES FOR NEXT PASS ***
        DO 675 I=1,N
        Q(I,J) = G(I,J)
  675   CONTINUE
  700   CONTINUE
  800   CONTINUE
C       *** END OF MAIN ITERATION LOOP ***
        STOP
        END
C
        SUBROUTINE IMESH(N,M,NOUT,ETA,H,RVEC)
        DIMENSION ETA(100),H(100),RVEC(10)
        REAL H
        WRITE(6,100)
  100   FORMAT(5X,'WHAT IS THE MAXIMUM EXTENT OF THE MESH?')
        READ(5,110) X
  110   FORMAT(F)
        SC = FLOAT(N/2)
        WRITE(6,130)
  130   FORMAT(5X,'HOW MANY POINTS OUTSIDE THE MESH?')
        READ(5,140) NOUT
  140   FORMAT(I)
        N = N + 2*NOUT + 2
        HT = X/SC
        ETA(1) = -(X+HT*NOUT)
        ETA(N) =  (X+HT*NOUT)
        DO 150 I=1,((N/2)-1)
        NI =  N - I
        H(I) = HT
        H(NI) = HT
        ETA(I+1) = ETA(I) + HT
        ETA(NI) = ETA(NI+1) - HT
  150   CONTINUE
C       WRITE(6,120) (ETA(I),I=1,N)
  120   FORMAT(5E15.6)
        RVEC(1) = 1.0
        RETURN
        END
C
        SUBROUTINE PZERO(N,M,ETA,RVEC,Q,KHOP,D)
        DIMENSION ETA(100),RVEC(10),Q(100,10),AL(2),CS(2)
        DIMENSION D(3),VAL(100)
        WRITE(6,10)
  10    FORMAT(5X,'ENTER EXPONENT AND CONSTANT FOR X<0')
        READ(5,20) AL(1),CS(1)
  20    FORMAT(2F)
        WRITE(6,30)
  30    FORMAT(5X,'ENTER EXPONENT AND CONSTANT FOR X>0')
        READ(5,20) AL(2),CS(2)
```

```
      WRITE(6,40)
   40 FORMAT(5X,'ENTER NUMBER OF INTERVALS TO CHOP')
      READ(5,50) KHOP
   50 FORMAT(I)
      WRITE(6,60)
   60 FORMAT(5X,'ENTER 1 FOR LINEAR, QUAD, CUBIC OR ZERO')
      READ(5,70) (D(J),J=1,3)
   70 FORMAT(3F)
      DO 300 J=1,M
C     *** INSERT RADIAL DEPENDENCE ONLY HERE ***
      CALL TAIL(N,ETA,N,ETA,AL,CS,KHOP,D,VAL)
      N2 = N/2
      DO 100 I=1,N2
      Q(I,J) = VAL(I)
  100 CONTINUE
      DO 200 I=N,N2+1,-1
      Q(I,J) = VAL(I)
  200 CONTINUE
  300 CONTINUE
      RETURN
      END
C
      SUBROUTINE MTEST(N,M,ETA,RVEC,Q,XK,AL,CS)
      DIMENSION ETA(100),RVEC(10),Q(100),XK(100),AL(10,2),CS(10,2)
      DO 140 J=1,M
      WRITE(6,110) ((AL(J,K),K=1,2),(CS(J,K),K=1,2))
  110 FORMAT(5X,4F12.5)
      WRITE(6,120) ETA(1),Q(1)
  120 FORMAT(//10X,F7.2,5X,E12.5)
      DO 140 I=2,N-1
      QP = 0.0
      IF(ABS(ETA(I)).LE. 1.E-3) GOTO 125
      IF(ETA(I).GT.0.) QP=AL(J,2)*CS(J,2)/ABS(ETA(I))**(AL(J,2)+1)
      IF(ETA(I).LT.0.) QP=AL(J,1)*CS(J,1)/ABS(ETA(I))**(AL(J,1)+1)
  125 WRITE(6,130) ETA(I),Q(I),XK(I),QP
  130 FORMAT(10X,F7.2,5X,E12.5,5X,E12.5,5X,E12.5)
  140 CONTINUE
      WRITE(6,150) ETA(N),Q(N)
  150 FORMAT(10X,F7.2,5X,E12.5)
      RETURN
      END
C
      SUBROUTINE PONE(N,M,ETA,RVEC,Q,XK,AL,CS)
      DIMENSION ETA(100),H(100),RVEC(10),Q(100,10),XK(100,10)
      DIMENSION AL(2),CS(2)
      WRITE(6,140)
  140 FORMAT(5X,'ENTER 1 FOR CUBIC TEST, 2 FOR TRUNCATION TEST')
      READ(5,150) NC
  150 FORMAT(I)
      IF( NC .NE. 2) GOTO 160
      WRITE(6,100)
  100 FORMAT(5X,'WHAT ARE THE ALPHAS AND CS <4F>?')
      READ(5,110) AL(1),AL(2),CS(1),CS(2)
  110 FORMAT(4F)
      DO 130 J=1,M
```

```
      DO 120 I=1,N
      X = ETA(I)
      Q(I,J) = X**4
      IF((I.EQ.1).OR.(I.EQ.N))GOTO 120
      XK(I,J) = 4.*X**3
  120 CONTINUE
  130 CONTINUE
      RETURN
  160 WRITE(6,170)
  170 FORMAT(5X,'ENTER THE CUBIC COEFFICIENTS <4F>.')
      READ(5,110) AL(1),AL(2),CS(1),CS(2)
      DO 180 J=1,M
      DO 180 I=1,N
      X = ETA(I)
      Q(I,J) = AL(1)+X*(AL(2)+X*(CS(1)+X*CS(2)))
      IF((I.EQ.1).OR.(I.EQ.N))GOTO 180
      XK(I,J)=AL(2)+X*(2.*CS(1)+3.*CS(2)*X)
  180 CONTINUE
      RETURN
      END
C
      SUBROUTINE TEST2(N,M,ETA,RVEC,Q,XK,G,AL,CS)
      DIMENSION ETA(100),RVEC(10),Q(100,10),XK(100,10),G(100,10)
      DIMENSION AL(10,2),CS(10,2)
1     DO 145 J=1,M
      WRITE(6,110) ((AL(J,K),K=1,2),(CS(J,K),K=1,2))
  110 FORMAT(5X,4F13.5)
      WRITE(6,120) ETA(1),Q(J,1)
  120 FORMAT(//7X,'1',2(5X,E13.5))
      DO 140 I=2,N-1
      WRITE(6,130) I,ETA(I),Q(I,J),XK(I,J),G(I,J)
  130 FORMAT(5X,I3,4(5X,E13.5))
  140 CONTINUE
  145 WRITE(6,150) N,ETA(N),Q(N,J)
  150 FORMAT(5X,I3,2(5X,E13.5))
      RETURN
      END
C
      SUBROUTINE STE T(N,M,NOUT,ETA,H,XK,Q,ALP,CF,KHOP,D)
      DIMENSION ETA(100),H(100),XK(100,10),Q(100,10)
      DIMENSION ALP(10,2),CF(10,2),D(3),VAL(100)
      DIMENSION AL(2),CS(2),AP(100),Z(100)
C     *** STATEMENT FUNCTION FOR EVALUATING SPLINE ***
      SEVAL(T) = Q(I+1,J)*T + Q(I,J)*T1 + H(I)*((XK(I,J)-D1)*T1
     1           - (XK(I+1,J)-D1)*T)*T*T1
C     *** INITIALIZE THE TEST STATISTICS ***
      DATA KMAX/10/
C     *** SET UP FOR RIGHT HAND SIDE OF MESH ***
      K = N/2
      KM1 = K - 1
      K1 = K + 1
C     *** SET UP THE LOOPS ***
      DO 60 J = 1,M
      AL(1) = ALP(J,1)
      AL(2) = ALP(J,2)
```

```
      CS(1) = CF(J,1)
      CS(2) = CF(J,2)
      DMAX = 0.0
      RMAX = 0.0
      XL2  = 0.0
C     *** GO OVER THE POINTS IN THE REFINED MESH ***
      XKMAX = FLOAT(KMAX)
      KOUNT = 0
      DO 50 I = K1,(N-NOUT-1)
      D1 = (Q(I+1,J)-Q(I,J))/H(I)
      DO 20 K = 1,KMAX
C     *** EVALUATE ON THE SUB-MESH POINTS ***
      T = FLOAT(K)/XKMAX
      T1 = 1.0 - T
      Z(K)  = ETA(I) + T*H(I)
      AP(K) = SEVAL(T)
   20 CONTINUE
      CALL TAIL(N,ETA,KMAX,Z,AL,CS,KHOP,D,VAL)
      DO 30 K=1,KMAX
C     *** UPDATE THE STATISTICS ***
      DEV = ABS(AP(K) - VAL(K))
      REL = DEV/ABS(VAL(K))
      DMAX = AMAX1(DMAX,DEV)
      RMAX = AMAX1(RMAX,REL)
      XL2  = XL2 + DEV*DEV
      KOUNT = KOUNT + 1
C     IF((I-K1).LE.3)WRITE(6,90) KOUNT,Z(K),AP(K),VAL(K),DEV,REL
   90 FORMAT(5X,I3,5E14.5)
   30 CONTINUE
   50 CONTINUE
      RMS = SQRT(XL2/FLOAT(KOUNT))
      WRITE(6,100) DMAX,RMAX,RMS
  100 FORMAT(5X,'The maximum error is ',e15.5/5x,
     1 'The maxium relative error is ',e15.5/5x,'The RMS is ',e15.5)
   60 CONTINUE
      RETURN
      END
C
      SUBROUTINE TAIL(N,ETA,M,Z,AL,CS,KHOP,D,VAL)
      DIMENSION Z(100),ETA(100),VAL(100),AL(2),CS(2),D(3)
C     *** STATEMENT FUNCTIONS FOR THE PROBLEM ***
      FUN(X,A,C) = C*ABS(X)**-A
      FACP(A,Y) = 1.+D(1)*A*Y*(1.+D(2)*(A+1)*Y*(1.+D(3)*(A+2.)
     1 *Y/3.)/2.)
      FACM(A,Y) = 1.-D(1)*A*Y*(1.-D(2)*(A+1.)*Y*(1.-D(3)*(A+2.)
     1 *Y/3.)/2.)
      N2 = N/2
C     *** EVALUATE THE FUNCTION ON THE GIVEN POINTS ***
      DO 100 I=1,M
      X = Z(I)
C     *** WHICH SIDE OF THE MESH IS THE POINT ON? ***
      IF( X .GT. 0 ) GOTO 50
C     *** DO CALCULATION FOR NEGATIVE SIDE ***
      C = CS(1)
      A = AL(1)
```

```
C .     *** CALCULATE POINT WHERE MATCHING IS DONE ***
        ESTAR = ETA(N2 - KHOP)
C       *** WHICH FORMULA IS USED? ***
        IF( X-ESTAR .LT. -0.0005 ) GOTO 25
        Y = (X/ESTAR) - 1.
        VAL(I) = FUN(ESTAR,A,C)*FACM(A,Y)
        GOTO 100
   25   VAL(I) = FUN(X,A,C)
        GOTO 100
C       *** DO CALCULATION ON POSITIVE SIDE ***
  50    C = CS(2)
        A = AL(2)
        ESTAR = ETA(N2+1+KHOP)
C       *** WHICH FORMULA IS USED? ***
        IF( X-ESTAR .GT. 0.0005) GOTO 75
        Y = (X/ESTAR) - 1.
        VAL(I) = FUN(ESTAR,A,C)*FACM(A,Y)
        GOTO 100
  75    VAL(I) = FUN(X,A,C)
  100   CONTINUE
C       if( N .eq. M )WRITE(6,110) (VAL(I),I=1,M)
  110   FORMAT(5X,5E15.5)
        RETURN
        END
```

```
      program test2
      dimension eta(100), f(100), b(4)
c     *** Statement Functions for the Evaluation of Integrals ***
      s0(g,x) = (cosh(z) - 1.)/g
      c0(g,x) = sinh(z)/g
      s1(g,x) = (y*cosh(z) + 1. - (c0(g,x)/es))/g
      c1(g,x) = (y*sinh(z) - (s0(g,x)/es))/g
      s2(g,x) = (y*y*cosh(z) - 1. - 2.*(c1(g,x)/es))/g
      c2(g,x) = (y*y*sinh(z) - 2.*(s1(g,x)/es))/g
      s3(g,x) = (y*y*y*cosh(z) - 3.*(c2(g,x)/es))/g
c     *** simple statement functions ***
      e0(g,x) = (exp(-z) - exp(-g*es))/g
      e1(g,x) = (y*exp(-z) + e0(g,x)/es)/g
      e2(g,x) = (y*y*exp(-z) + 2.*e1(g,x)/es)/g
      e3(g,x) = (y*y*y*exp(-z) + 3.*e2(g,x)/es)/g
c     *** definitions for the far field contribution ***
      ft1(x,ia,g) = (g**(ia-1))*ei(ia,g,x)
      ft2(x,ia,g) = sinhi(ia,del,z1)/(xmesh**(ia-1))
c     *** ask user for the size of mesh ***
      write(6,100)
  100 format(5x,'What are the number of intervals?')
      read(5,*) n
      write(6,110)
  110 format(5x,'What is the extent of the mesh?')
      read(5,*) x
      sc = float((n/2)-2)
      ht = x/sc
      eta(1) = -(x+ht)
      eta(n) = x+ht
      do 10 i=2,n/2
      ni = 1 + n - i
      eta(i) = eta(i-1) + ht
      eta(ni) = eta(ni+1) - ht
   10 continue
      xm = eta(n-1)
      write(6,130)
  130 format(5x,'What is the value of F?')
      read(5,*) f1
   12 write(6,140)
  140 format(5x,'What is the absolute value of the rate of decay?')
      read(5,*) ia
      write(6,150)
  150 format(5x,'How many intervals are chopped?')
      read(5,*) khop
      if( khop .lt. 1 ) goto 12
      if( ia .lt. 1 ) goto 12
c     *** Code for various degrees of osculation ***
   13 write(6,160)
  160 format(5x,'What is the degree of osculation (0-3) ?')
      read(5,*) id
      if( (id .lt. 0) .or. (id .gt. 3) ) goto 13
      do 14 i=1,4
      b(i) = 0.0
      if( (i-1) .le. id ) b(i) = 1.0
   14 continue
```

```
      n2 = n/2
      fm = f1*eta(n-1)
      es = eta(n2+khop+1)
      z1 = f1 * es
      xmesh = es
      a = float(ia)
c     *** do positive side of the mesh for comparison ***
      do 20 i=n2+1,n
      x = eta(i)
      z = f1*x
      y = (x/es) - 1.
      c = exp(-z)
      d = sinh(z)
      if( i .gt. n2+1+khop) goto 15
      p1 = b(1)*s0(f1,x) - a*(b(2)*s1(f1,x) - .5*(a+1.)*
     1 (b(3)*s2(f1,x) - (a+2.)*b(4)*s3(f1,x)/3.))
      p2 = b(1)*e0(f1,x) - a*(b(2)*e1(f1,x) - .5*(a+1.)*
     1 (b(3)*e2(f1,x) - (a+2.)*b(4)*e3(f1,x)/3.))
      f(i) = (p1*c + p2*d)/(es**ia) + d*ft1(es,ia,f1)
      goto 20
  15  del = (x-xmesh)/xmesh
      z = f1*es
      y = 0.
      p1 = b(1)*s0(f1,es) - a*(b(2)*s1(f1,es) - .5*(a+1.)*
     1 (b(3)*s2(f1,es) - (a+2.)*b(4)*s3(f1,es)/3.))
      f(i) = (p1/(es**ia) + ft2(x,ia,f1))*c + ft1(x,ia,f1)*d
  20  continue
c     *** now that the expressions have been evaluated, print 'em! ***
      do 30 i=n2+1,n
      write(6,170) i,eta(i),f(i)
 170  format(5x,i4,6e14.6)
  30  continue
      write(6,180)
 180  format(/5x,'Table for Checking Integration Algorithm'/)
      stop
      end
c
      function ei(ia,f,x)
c     *** function for the evaluation of the second integral ***
c     *** test for special case when arguement is zero ***
      if( abs( f*x ) .ge. 1.e-05) goto 5
      ei = 1./float(ia-1)
      return
c     *** initialize starting values ***
    5 fs = 0.
      fm = -1.*f
      fx = fm * x
      c = 1.0
      if( ia .eq. 1 ) goto 20
c     *** do integration by parts until e sub 1 is reached ***
      do 10 i=ia,2,-1
      i1 = i - 1
      p = float(i1)
      fs = fs + (c/p)*exp(fx)/x**i1
      c = c * fm/p
```

```
   10   continue
c       *** calculate e/sub 1/ using continued fraction approximation
c           from page 231 of Abramowitz & Stegun ***
   20   fx = -1.*fx
        e1 = 1.039549/(1.43679 + fx)
        e1 = 0.99592/(1.893888 + fx - e1 )
        e1 = (1. - e1)* exp(-fx)/fx
c       *** final step ***
        fs = fs + c*e1
        ei = fs
        return
        end
c
        function sinhi(ia,d,xnu)
        external func, funs
c       *** initialize starting values ***
        f = 0.
        c = 1.
        iflag = 1
        x = 1. + d
        if( x*xnu .ge. 85. ) write(6,100) ia,d,xnu
  100   format(5x,'ERROR--Overflow for combination ',i5,2e15.7)
        cu = cosh(x*xnu)
        cl = cosh(xnu)
        su = sinh(x*xnu)
        sl = sinh(xnu)
        if( ia .eq. 1 ) goto 32
c       *** do the integration by parts until base value is reached ***
        do 30 i=ia,2,-1
        i1 = i - 1
        p = float(i1)
        if( iflag .eq. 0 ) goto 15
        fu = su/x**i1
        fl = sl
        iflag = 0
        goto 20
   15   fu = cu/x**i1
        fl = cl
        iflag = 1
   20   f = f + (c/p)*(fl-fu)
        c = c * (xnu/p)
   30   continue
c       *** insert code here for shi(x) - shi(1.) calculation
   32   xone = 1.0
        tol = 1.e-04
c       *** current code uses Romber extrapolation for integral ***
        if( iflag .eq. 0 ) goto 35
        call romb(funs, xone, x, tol, val, xnu)
        f = f + c*val
        goto 40
   35   call romb(func, xone, x, tol, val, xnu)
        f = f + c*val
   40   continue
        sinhi = f
        return
```

```
      end
c
      subroutine func(x,n,fval,xm)
      dimension x(256), fval(256)
      do 10 i=1,n
      z = x(i)
      fval(i) = cosh(z*xm)/z
   10 continue
      return
      end
c
      subroutine funs(x,n,fval,xm)
      dimension x(256), fval(256)
      do 10 i=1,n
      z = x(i)
      fval(i) = sinh(z*xm)/z
   10 continue
      return
      end
c
      subroutine romb(funct, xlow, xup, tol, val, xm)
c     *** this subroutine calculates a definite integral using the ***
c     *** extrapolation scheme based on the trapezoidal rule ***
      dimension x(256), fval(256), y(10,10)
      external funct
      do 10 i=1,10
      do 10 j=1,10
   10 y(i,j) = 0.0
      sum = 0.0
      x(1) = xlow
      x(2) = (xlow + xup)/2.
      x(3) = xup
      call funct(x,3,fval,xm)
      sum = sum + fval(1) + fval(3) + 2*fval(2)
      step = (xup - xlow)/2.
      y(1,1) = .5* step * sum
      do 50 i=1,7
      step = step/2.
      do 20 j=1,2**i
   20 x(j) = xlow + (-1+2*j)*step
      call funct(x,2**i,fval,xm)
      temp = 0.0
      do 30 j=1,2**i
   30 temp = temp + fval(j)
      sum = sum + 2.*temp
      y(1,i+1) = .5 * step * sum
      n = 0
      do 40 j=2,i+1
      n = 2 + n
   40 y(j,i+1) = ((y(j-1,i+1)*2**N)-y(j-1,i))/float(-1+2**n)
      if( abs(y(i+1,i+1)-y(i,i+1)) .le. (tol*y(i,i+1)))
     1 goto 60
   50 continue
c     write(6,100)
  100 format(5x,'Warning, Romberg Extrapolation failed; last value used')
```

```
      i = i - 1
 60   val = y( i+1, i+1 )
      return
      end
```

END

8-87

DTIC